Etude de quelques organisations d'antémémoires

Nathalie Drach, André Seznec

Rapports de Recherche

1992

3ème anniversaire

N° 1775

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

ÉTUDE DE QUELQUES ORGANISATIONS D'ANTÉMÉMOIRES

Nathalie DRACH
André SEZNEC

Octobre 1992

RR_1775

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Etude de quelques Organisations d'Antémémoires

Nathalie Drach, André Seznec
Programme 1
Projet CALCPAR *

5 octobre 1992

Publication Interne n° 678 - 44 pages

Résumé

Les performances des systèmes batis atour d'un microprocesseur dépendent de plus en plus des performances de la hiérarchie mémoire, et plus particulièrement des antémémoires. En effet depuis ces dernières années, le temps de cycle du processeur a diminué beaucoup plus vite que le temps d'accès à la mémoire principale. Cette tendance a augmenté l'importance des antémémoires et de leur efficacité. Dans ce rapport, nous présentons trois nouvelles organisations d'antémémoires situées sur la même puce que le processeur et qui maximisent les performances des microprocesseurs. La première organisation est dérivée de l'organisation d'antémémoire unifiée à laquelle on a ajouté un tampon instructions. La deuxième organisation est composée d'une antémémoire unifiée instructions et données et d'une antémémoire séparée instrutions. Enfin nous présentons l'organisation d'antémémoires semi-unifiées. Cette organisation est composée de deux antémémoires, C_1 et C_2, séparées physiquement, mais destinées à recevoir instructions et données. L'antémémoire C_1 est à la fois antémémoire principale pour les instructions et antémémoire secondaire pour les données (C_2 est antémémoire principale pour les données et antémémoire secondaire pour les instructions). Ainsi le degré d'associativité pour les données et les instructions est artificiellement augmenté et l'espace de stockage se répartit dynamiquement entre instructions et données. Les antémémoires semi-unifiées diminuent le taux de défaut d'antémémoires par rapport aux organisations d'antémémoires couramment utilisées dans les microprocesseurs.

Performance evaluation of some cache organizations

Abstract

Today microprocessor systems performance mainly depends on the performance of their memory hierarchy, and particularly on the performance of small on-chip caches. In this paper, we present three new organizations of on-chip caches, and particularly the semi-unified caches organization. This organization consists of two physically split caches C_1 and C_2; both C_1 and C_2 are used for storing instructions and data. Cache C_1 is, at the same time the main cache for instructions and the secondary cache for data (C_2 is the main cache for data and secondary cache for instructions). Thus for instructions and data, the degree of associativity is artificially increased and the cache space respectively devoted to each type is dynamically adjusted. Semi-unified caches typically exhibit lower miss ratios than split caches.

*Ce travail a été partiellement soutenu par le PRC-Architecture de Machines Nouvelles

Table des matières

1 Introduction

En raison de l'écart grandissant entre temps de cycle du processeur et temps d'accès à la mémoire principale, l'accès aux données est souvent le facteur limitatif des performances du processeur. L'ajout de mémoires plus rapides (donc plus chères) et de petites tailles entre le processeur et la mémoire principale permet de donner l'illusion que le temps moyen d'accès aux données est relativement faible. Ce concept architectural est appelé hiérarchie mémoire [PAT90].

Un niveau de mémoire intermédiaire est appelé antémémoire. Compte tenu de l'évolution technologique des circuits intégrés [KOHN89], il est devenu possible d'avoir sur une même puce le processeur et une petite antémémoire; c'est le cas de la plupart des nouvelles architectures des microprocesseurs RISC ([DEC21064], [i860], [MIPS R4000], [superSPARC]). Cette évolution se concrétise par une augmentation du nombre de transistors sur une puce et par une augmentation de la vitesse des composants (25 % tous les ans). Beaucoup de concepteurs adjoignent une antémémoire secondaire entre la mémoire principale et la puce du processeur. Dans le cadre de notre étude nous considérons un seul niveau d'antémémoire placé sur la puce du processeur.

L'organisation des antémémoires joue un rôle important dans l'augmentation des performances. Cette organisation d'antémémoires dépend de plusieurs paramètres: espace de la puce du processeur dédié à l'antémémoire, partition de cet espace en fonction d'un type (instruction ou donnée), taille de l'élément de base utilisé pour les transferts, mécanisme de placement dans l'antémémoire, stratégie de remplacement et politiques de recopie en mémoire [PRZY88], [PUZAK85], [SMITH82]. Ainsi, le but de notre étude est d'essayer d'organiser les antémémoires afin d'en maximiser les performances. Nous mettons l'accent sur l'organisation d'antémémoires semi-unifiées qui au vu de nombreuses expérimentations devrait améliorer sensiblement les performances des microprocesseurs. Dans la section 2, nous rappelons les caractéristiques essentielles de la hiérarchie mémoire et de ses différents éléments: la mémoire principale et l'antémémoire et nous détaillons les principales organisations d'antémémoires. Puis dans la section 3, nous introduisons de nouvelles organisations d'antémémoires et en particulier les antémémoires semi-unifiées. Dans la section 4, nous faisons l'étude comparative des performances des différentes organisations.

2 Hiérarchie mémoire

2.1 Notion de hiérarchie mémoire

2.1.1 Définition d'une hiérarchie mémoire

- **Le principe de la hiérarchie mémoire:**

  Le rôle d'une hiérarchie mémoire est de diminuer les temps moyens d'accès du processeur aux instructions et données [PRZY90].

  Deux principes essentiels sont à l'origine de ce concept:

  - Le coût par bit d'une mémoire croit lorsque son temps d'accès décroit. D'autre part le coût global d'une mémoire est proportionnel à sa taille. Par conséquent il est très coûteux de réaliser une mémoire de grande taille et très rapide. Aussi plus une mémoire devra être rapide, plus sa taille sera réduite.
  - Les programmes ont en général des propriétés de localité temporelle (un élément référencé à t_0 a une forte probabilité d'être référencé à $t + t_0$ avec t petit) et de localité spatiale (si l'élément situé à

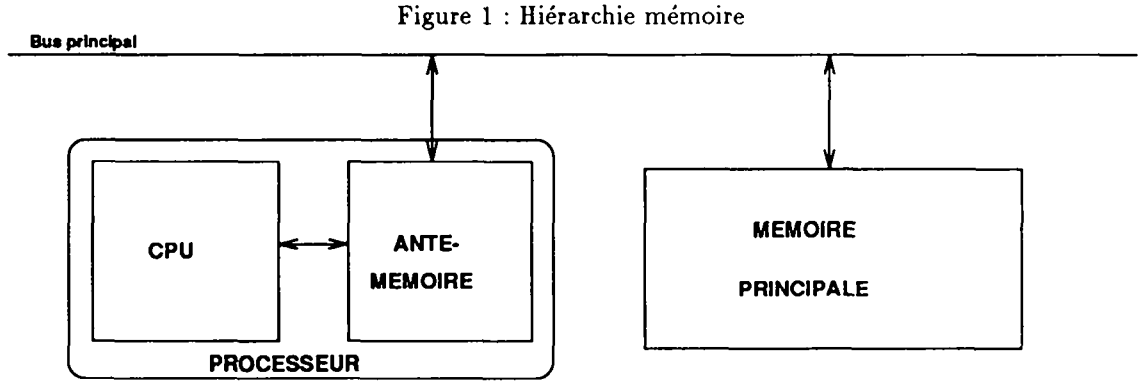
l'adresse x_0 est référencé, l'élément situé à $x_0 + x$ avec x petit a une forte probabilité d'être référencé dans un avenir proche).

La hiérarchie mémoire est un agencement de plusieurs mémoires ayant différentes caractéristiques: temps d'accès, capacités et coûts. Au niveau supérieur, proche du CPU, de petites mémoires très rapides sont utilisées. Elles sont destinées à stocker les éléments récemment référencés. Et plus l'éloignement du CPU est grand, plus les mémoires sont grosses et lentes. Le principe de cette hiérarchie est que toute donnée appartenant au niveau mémoire N appartient à tous les niveaux supérieurs $N-1$, ..., 1 en prenant comme niveau N la mémoire la plus lente.

2.1.2 Hiérarchie mémoire étudiée

Pour notre étude, la hiérarchie mémoire se compose d'une mémoire principale (niveau 2) et d'un niveau de mémoire (niveau 1) situé sur la puce du processeur (figure 1). Le niveau 2 est une grosse mémoire avec un temps d'accès relativement lent et un coût par bit modéré. Le niveau 1 est une mémoire de petite capacité avec une grande rapidité d'accès, mais avec un coût par bit élevé. Dans les sections 2.2 et 2.3, nous détaillerons les caractéristiques de la mémoire principale et de l'antémémoire qui seront utiles pour nos différentes études.

Figure 1 : Hiérarchie mémoire



2.1.3 Evaluation des performances d'une hiérarchie mémoire

Le but d'une hiérarchie mémoire étant de diminuer le temps moyen d'accès aux éléments et par suite le temps d'exécution T_{exec} d'un programme P, nous allons utiliser T_{exec} pour caractériser les performances des organisations étudiées. Pour simplifier l'étude, nous considérons le cas des microprocesseurs RISC séquençant une instrution par cycle (par exemple le MIPS R4000). Biensur nous travaillons dans le cadre des architectures pipelinées (la technique du pipeline est illustrée figure 10).

$$T_{exec} = \textit{Temps d'exécution des instructions} + \textit{Temps de lecture des instructions et des données}$$

- Temps d'exécution des instructions:

Dans quelques architectures RISC (Reduced Instruction Set Computer) [ETIEM91], certaines instructions (Load, Store, etc...) requièrent plus d'un cycle; c'est le cas des premières versions du Sparc. Pour notre étude, nous allons supposer que chaque instruction s'exécute en un cycle.

Temps d'exécution des instructions $= N_{inst} * T_{CPU}$

- $T_{CPU} = 1$ cycle

- Temps de lecture des instructions et des données:

  Temps de lecture des instructions et des données $= N_{ref} * \lambda_{miss} * T_2 + N_{ref} * \lambda_{hit} * T_1$

  Dans le cas où il existe deux chemins de données entre l'espace de stockage sur la puce du processeur et le processeur lui-même, la technique du pipeline donne l'illusion que les phases d'exécution et de chargement des instructions et des données se font en parallèle (en fait, elles sont décalées d'un certain nombre de cycles égal au nombre d'étages du pipeline). Par conséquent on peut approximativement considérer que le temps d'exécution des instructions représente également le temps de chargement des données et des instructions présentes dans les antémémoires (dans la mesure où les chargements peuvent s'effectuer en un cycle).

  Ainsi le *temps de lecture des instructions et des données* se réduit à:

  Temps de lecture des instructions et des données $= N_{ref} * \lambda_{miss} * T_2$

- Notations:

  - T_{exec} : temps d'exécution d'un programme P.
  - N_{inst} : nombre d'instructions de P.
  - N_{don}: nombre de données de P.
  - N_{ref}: $N_{ref} = N_{inst} + N_{don}$
  - T_{CPU} : temps d'exécution d'une instruction par le CPU.
  - λ_{miss}: fraction de données et d'instructions n'appartenant pas au niveau 1 par rapport au nombre total N_{ref}: $\lambda_{miss} = \lambda_{miss}^{donnees} + \lambda_{miss}^{inst}$
  - λ_{hit}: fraction de données et d'instructions appartenant au niveau 1 par rapport au nombre total N_{ref}: $\lambda_{hit} = \lambda_{hit}^{donnees} + \lambda_{hit}^{inst}$
  - T_1: temps nécessaire pour accéder à une requête présente au niveau supérieur (niveau 1).
  - T_2: temps nécessaire pour aller chercher dans les niveaux inférieurs la requête en cours. Ce temps T_2 sera étudié plus en détail dans la section 2.2.

On a par conséquent sur les processeurs RISC utilisant la technique du pipeline:

- si les accès aux données et aux instructions ne peuvent pas se faire en parallèle:

$$T_{exec} = N_{inst} * T_{CPU} + N_{ref} * (\lambda_{miss} * T_2 + \lambda_{hit}^{donnees} * T_1) \quad (1)$$

- si les accès aux données et aux instructions peuvent se faire en parallèle:

$$T_{exec} = N_{inst} * T_{CPU} + N_{ref} * (\lambda_{miss} * T_2) \quad (2)$$

A présent déterminons les paramètres de T_{exec} qui ne dépendent pas de l'organisation des antémémoires. Ainsi dans l'étude des performances (étude comparative) nous pourrons nous consacrer aux seuls paramètres influents:

$$T_{exec} = f(N_{inst}, N_{don}, T_{CPU}, \lambda_{miss}, \lambda_{hit}, T_1, T_2) \quad (3)$$

- N_{inst} et N_{don} sont fixes pour P quelque soit la hiérarchie mémoire utilisée.
- T_{CPU} n'est pas un paramètre de la hiérarchie mémoire. Il dépend du CPU.

- **Remarque:** l'unité de temps est le cycle CPU.

2.2 Mémoire principale

Ce niveau de mémoire est indispensable au bon fonctionnement de la hiérarchie mémoire du fait de sa forte capacité de stockage à moindre coût.

La mémoire principale contient les instructions et les données du programme à exécuter, ainsi qu'une partie du système d'exploitation. Pour exécuter un programme chargé en mémoire principale, les instructions et les données sont lues dans la mémoire principale au fur et à mesure de leur utilisation et envoyées au processeur. Après avoir lancé une requête à la mémoire principale (ex: lecture de données), le processeur doit attendre un certain nombre de cycles appelé temps d'accès T_2 avant que le résultat de cette requête soit valide (c-à-d utilisable par le processeur). Les transferts entre la mémoire principale et le processeur se font par lignes. Une ligne est un ensemble de plusieurs mots. Le temps nécessaire pour accéder au premier mot d'une ligne est appelé latence mémoire. Ensuite les autres mots de la ligne sont lus consécutivement.

$$T_2 = T_{lat} + t(mot) \quad (4)$$

- T_{lat}: nombre de cycles nécessaire pour que le premier mot d'une ligne lue en mémoire soit transféré au processeur (temps de réponse de la mémoire).
- $t(mot)$: après le temps T_{lat} les mots de la ligne lue peuvent être disponibles, mais ils ne sont pas tous accessibles par le processeur au même moment. $t(mot)$ est le nombre de cycles après le temps T_{lat} pour que *mot* soit effectivement disponible. Ce paramètre dépend de la largeur du bus entre le processeur et la mémoire principale: il correspond au débit de la mémoire. L'ordre dans lequel les mots d'une ligne sont envoyés au CPU, peut varier. Sur certains microprocesseurs, c'est le mot demandé qui arrive en premier au CPU et non le premier mot de la ligne à laquelle il appartient, alors $T_2 = T_{lat}$.

- **Remarque**: La puissance d'un ordinateur dépend essentiellement de la vitesse d'accès à la mémoire principale (fonction de la latence mémoire) et de sa capacité. En effet depuis 1985 les performances du processeur augmentent de 50 à 100% par an alors que celles des DRAM (Dynamic Random Access Memory), componsants des mémoires principales actuelles, est de 7% [PAT90]. La latence mémoire est donc un facteur critique conditionnant les performances d'un ordinateur.

2.3 Antémémoire

Le principe de l'antémémoire apporte une solution à la différence de temps de cycles entre le CPU et la mémoire principale. Entre le CPU et la mémoire principale, une petite mémoire très rapide est insérée. Elle va contenir

des instructions et des données requises par le CPU. En 1992 cette antémémoire se trouve généralement sur la même puce que le CPU. Elle fonctionne dans la plupart des cas à la même vitesse que le processeur [MATI89]. L'espace de stockage disponible pour l'antémémoire est la principale contrainte intervenant dans la conception de ces antémémoires. Le principe et le succès du fonctionnement des antémémoires reposent sur les propriétés de localité des programmes (voir 2.1.1).

2.3.1 Fonctionnement de l'antémémoire

L'antémémoire est structurée par lignes, mais le CPU peut accéder à l'antémémoire mot par mot.

Soit $E(t)$ l'ensemble des données et des instructions présentes dans l'antémémoire à l'instant t. Le processeur envoie une requête (référence mémoire) REF à l'antémémoire à t, alors:

- soit $REF \in E(t)$: c'est un succès ou *hit*. La requête suivante peut alors être traitée.
- soit $REF \notin E(t)$: c'est un échec appelé encore défaut d'antémémoires ou *miss* et la requête est alors transmise à la mémoire principale. Une fois la référence lue en mémoire, elle est copiée dans l'antémémoire et envoyée au CPU. Le mécanisme de placement de la donnée dans l'antémémoire dépend du degré d'associativité A et de la stratégie de remplacement S.

2.3.2 Associativité de l'antémémoire

Ce mécanisme de placement dans l'antémémoire a fait l'objet de nombreuses études [AGAR89], [HILL88], [HILL89], [PRZY90].

Supposons qu'on ait une antémémoire de N lignes. Le nombre de lignes de l'antémémoire où peut être placée une nouvelle donnée REF est lié au degré d'associativité A de l'antémémoire.

- $A = 1$: REF ne peut être chargée dans l'antémémoire qu'à une seule place. Cette antémémoire est à correspondance directe.
- $A = N$: REF peut être chargée à n'importe quelle place. Cette antémémoire est dite totalement associative.
- $2 \leq A \leq N/2$: REF peut être chargée dans A places de l'antémémoire. Ces A places appartiennent à A ensembles distincts appelés bancs. Elles sont obtenues par un calcul de modulo (en général (numéro de la ligne mémoire) *modulo* (taille des bancs)). Parmi celles-ci, la ligne à remplacer est choisie selon la stratégie S. Cette antémémoire est dite associative par ensembles de A lignes ou *A-ways set-associative*.

2.3.3 Stratégies de remplacement

Les stratégies de remplacement les plus utilisées sont:

- la politique de choix aléatoire (Random): elle consiste à choisir la ligne cible aléatoirement. Cette politique est très peu chère mais également peu fiable du point de vue des performances.
- la politique LRU (Least Recently Used): elle consiste à choisir comme ligne cible la ligne la plus anciennement référencée. Cette politique donne de bons résultats, mais elle est assez coûteuse dès que l'associativité croit [SMITH82].

2.3.4 Politiques d'écriture

Le nombre d'accès dans l'antémémoire en lecture domine sur le nombre d'accès en écriture (les accès aux instructions sont toujours des accès en lecture). Mais les accès en lecture ne posent pas de réels problèmes, tandis que les accès en écriture demandent une gestion fiable de la cohérence des données (c-à-d la mise à jour lors d'une modification). Les deux politiques d'écriture de base [SMITH82] sont *la recopie* et *l'écriture simultanée*.

Après avoir modifié une ligne dans l'antémémoire:

- soit cette ligne est écrite tout de suite en mémoire principale (écriture simultanée ou *Write-through*).
- soit cette ligne est marquée 'modifiée' et sera réecrite en mémoire lorsqu'elle sera cible d'un remplacement (politique de recopie ou *Write-back*).

Table 1 : Principales caractéristiques des antémémoires pour certains processeurs RISC.

Caractéristiques	super SPARC	DEC alpha	MIPS R4000	Intel i860	IBM RS 6000
Année	1992	1992	1992	1989	1990
Antémémoire instructions (Ko)	20	8	8	4	8
Antémémoire données	16	8	8	8	64
Appartient à la puce du CPU	oui	oui	oui	oui	non
Taille de la ligne (o)	64 (inst.) 32 (don.)	32	16 à 32	32	64 (inst.) 128 (don.)
Associativité	5-way (inst) 4-way (don.)	directe	directe	2-way	2-way (inst.) 4-way (don.)
Politique d'écriture	simult. recopie	simult	recopie	recopie	recopie
Bus inst. (bits)	64	64	64	64	64
Bus don. (bits)	64	64	64	128	128/64
Remplacement	pseudo LRU	-	-	Random	LRU

Le tableau 1 présente pour plusieurs processeurs RISC (Reduce Instruction Set Computer) les caratéristiques physiques des antémémoires: taille des antémémoires, taille de la ligne, degré d'associativité, politique d'écriture, taille des différents bus et position des antémémoires par rapport à la puce du CPU. Les différents processeurs présentés utilisent deux antémémoires instructions et donnée séparées.

2.4 Organisations actuelles d'antémémoires

Les organisations d'antémémoires actuellement utilisées dans les microprocesseurs sont de l'un des deux types suivants:

1. **Antémémoires instructions et données séparées**

Cette organisation est la plus couremment utilisée (Harvard Organisation figure 2). Le processeur contient deux antémémoires distinctes qui stockent respectivement les instructions et les données. Chacune des ces antémémoires est reliée à l'UC (Unité de Contrôle) par un bus.

2. **Antémémoire unifiée**

Des antémémoires unifiées ont été utilisées (figure 3) dans la plupart des versions du SPARC. Jusqu'en 1991, la plupart des antémémoires situées sur la puce du processeur sont séparées, mais le Power PC d'IBM/Motorola va utiliser le principe d'unification pour son antémémoire *on-chip*. Le processeur contient une antémémoire stockant indifféremment les instructions et les données. Cette antémémoire est reliée à l'UC par un bus.

3. **Avantages et inconvénients de ces organisations**

En comparant ces deux organisations, certains avantages et inconvénients apparaissent récapitulés dans le tableau ci-dessous. Cette liste n'est pas exhaustive, elle nous permet seulement de mettre en évidence les inconvénients auxquels nous allons tenter d'apporter une solution.

	Antémémoires séparées: O_{separe}	Antémémoires unifiées: O_{unifie}
AVANTAGES	- la bande passante est doublée entre l'UC et les antémémoires (*) - les conflits entre les données et les instructions sont supprimés	- l'espace de stockage pour les données et les instructions est variable, fonction des besoins de chacun
INCONVENIENTS	- l'espace attribué à chaque catégorie d'éléments est fixe: en général la moitié de la place totale	- il n'y a qu'un seul bus entre l'UC et l'antémémoire - aucune place minimale n'est présevée pour les instructions ou les données

* Les données pourront transiter en même temps qu'une instruction (leurs transferts demanderont aucun cycle).

Solutions

- Afin de remédier aux inconvénients des antémémoires unifiées et en particulier afin de séquencer simultanément les instructions et les données (voir justification 3.1), nous allons apporter des modifications architecturales tout en préservant le principe d'unification. Ces modifications sont concrétisées par l'ajout d'un tampon instructions en dérivation entre l'antémémoire unifiée et l'UC.
- Mais nous allons également présenter deux organisations d'antémémoires basées sur le principe de séparation physique des antémémoires et d'unification instructions/données. Ces deux organisations palient aux inconvénients présentés pour les organisations d'antémémoires séparées et unifiées tout en essayant de préserver leurs avantages. Ceux sont l'organisation d'antémémoires mixtes et l'organisation d'antémémoires **semi-unifiées**. Elles ont été conçues afin d'optimiser l'utilisation de l'espace

mémoire en fonction des besoins en données et en instructions. La première organisation, antémémoires mixtes, utilise la localité spatiale des instructions et permet de bénéficier d'un espace de stockage pour les données beaucoup plus important que dans le cas des antémémoires séparées.

La deuxième organisation, antémémoires semi-unifiées, permet de gérer l'espace nécessaire en fonction des besoins de chaque type (instruction ou donnée). Mais cette gestion est menée de telle façon qu'elle n'engendre pas des situations extrêmes comme dans le cas des antémémoires unifiées. De plus elle permet d'éviter certains phénomènes parasites des antémémoires mixtes.

Figure 2 : Antémémoires séparées

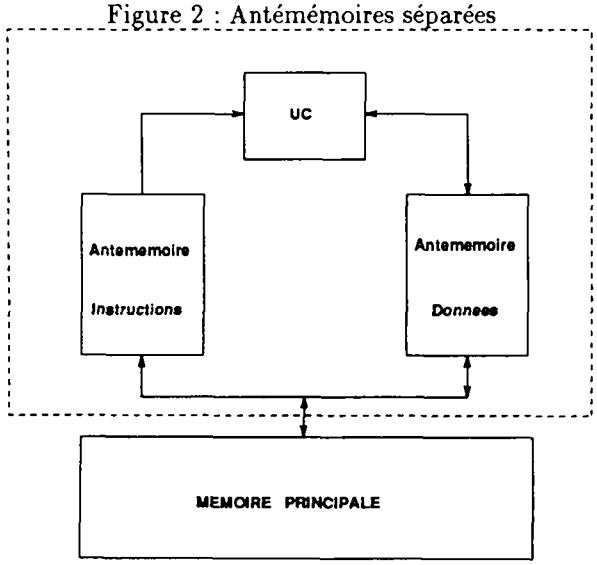
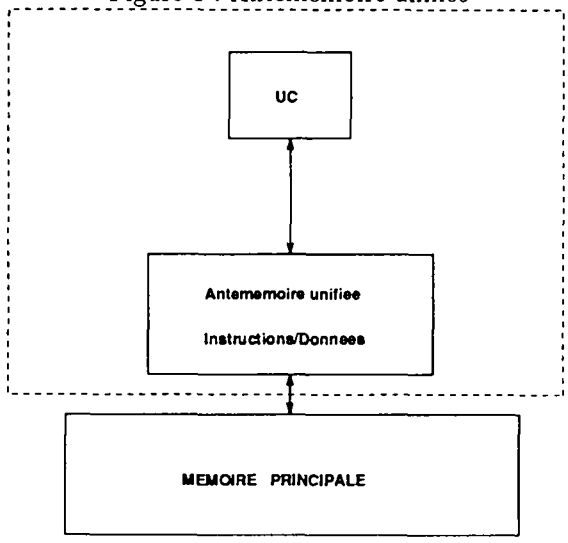


Figure 3 : Antémémoire unifiée



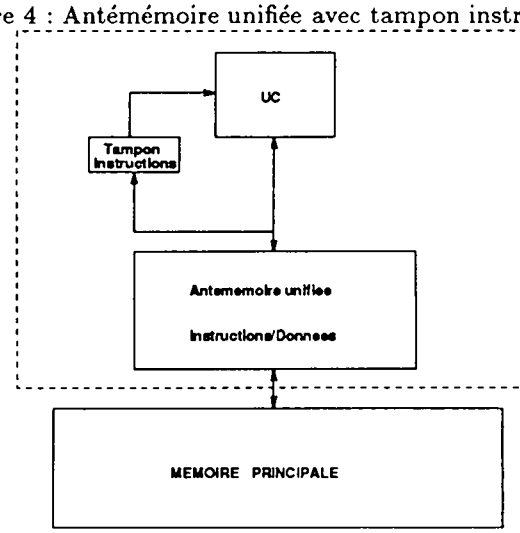
3 Recherche de l'organisation d'antémémoires optimale

Nous allons proposer trois organisations d'antémémoires succeptibles de maximiser les performances d'un microprocesseur: les antémémoires unifiées avec tampon instructions, les antémémoires mixtes et surtout les antémémoires semi-unifiées.

3.1 Antémémoire unifiée avec tampon instructions

Cette organisation architecturale est constituée d'une antémémoire unifiée données et instructions et d'un tampon (figure 4) destiné à ne recevoir que des éléments de type instruction [HILL87], [FARR89]. L'utilisation de ce tampon instructions est liée aux propriétés de localité spatiale et aux propriétés des circuits intégrés. En effet le flot d'instructions présente une localité spatiale importante. Et depuis que l'antémémoire principale est sur la même puce que l'UC, il est devenu possible de transférer toute une ligne d'instructions à chaque cycle entre les différents composants de la puce (donc de l'antémémoire unifiée vers le tampon instructions). L'accès aux instructions ne sature donc plus l'antémémoire laissant ainsi des cycles disponibles pour l'accès aux données. *Il semble que le processeur PowerPC utilisera une architecture de ce type.*

Figure 4 : Antémémoire unifiée avec tampon instructions



Nous allons donner les caractéristiques spécifiques mises en oeuvre pour l'antémémoire unifiée et le tampon instructions. En ce qui concerne l'antémémoire unifiée, nous présentons une stratégie de remplacement appelée stratégie OL (Old Lines) qui est plus adaptée à cette organisation que la stratégie LRU. De plus la mise en oeuvre matérielle de la stratégie OL est facile et peu coûteuse. Les caractéristiques présentées pour le tampon instructions sont éssentiellement liées au mécanisme de remplissage des lignes de ce tampon.

- **Antémémoire unifiée:**

  Nous avons mis en oeuvre une stratégie de remplacement pour l'antémémoire unifiée. Cette stratégie, appelée stratégie OL (Old Line), opère de la manière suivante. Soit L_{miss} une ligne ayant provoqué un

miss instruction. Soit $\mathcal{E}(L_{miss})$, l'ensemble des lignes pouvant recevoir L_{miss} ($card(\mathcal{E}(L_{miss})) = A_{unifie}$). $\mathcal{E}(L_{miss})$ est composé de deux sous-ensembles: le sous-ensemble $V(L_{miss})$ contenant les lignes dites *anciennes* et le sous-ensemble $R(L_{miss})$ contenant les lignes les plus récentes et tel que $V \bigcup R = \mathcal{E}$. Une ligne est *ancienne* si elle appartient à l'antémémoire depuis un certain temps (en nombre de cycles) sans qu'on y ait accédé.

- $card(V(L_{miss})) \neq 0$: une des lignes *anciennes* est choisie au hasard comme ligne cible pour L_{miss}.
- $card(V(L_{miss})) = 0$: une ligne cible est choisie dans $R(L_{miss})$ selon une politique LRU. Mais cette ligne peut être choisie en priorité dans R_{inst} où $R = R_{inst} \bigcup R_{don}$.

Remarques:

1. La stratégie OL s'applique de la même manière aux miss sur une donnée (remplacer *instruction* par *donnée*).
2. Différentiation entre données et instructions:

   Nous avons fait des études sur plusieurs stratégies de remplacement et en particulier sur la stratégie LRU et la stratégie OL. Ces deux stratégies donnent de meilleurs résultats lorsque les données et les instructions sont différenciées. En effet soit $\mathcal{E} = \mathcal{E}_{don} \bigcup \mathcal{E}_{inst}$, si lors d'un miss de type i ($i \in \{$ *donnée*, *instruction* $\}$), la stratégie de remplacement utilisée est appliquée seulement à $\mathcal{E}_i$, les performances sont améliorées.
3. Pour éviter des cas extêmes, on peut imposer qu'un type déléments n'occupe pas une fraction de l'antémémoire inférieure ou supérieure à des seuils donnés. Pour ce faire, on peut compter le nombre de lignes de données et de lignes d'instructions dans l'antémémoire.
4. Cette stratégie est facile à implémenter du point de vue matériel, il suffit d'utiliser ou de rajouter des tags pour chaque ligne de l'antémémoire unifiée:

   1 bit pour savoir si une donnée est *ancienne* ou pas (remis à jour tous les X cycles)

   1 bit pour savoir si une ligne contient des données ou des instructions

   Le côut de cette mise en oeuvre est tout à fait raisonnable. Si $L = 64$ o, l'espace alloué pour la mise en oeuvre de cette stratégie représente 0.4% ($2/L$ %) de l'espace mémoire total E_{ant}.

- **Tampon instructions:**

Nous avons élaboré un mécanisme afin de remplir les lignes du tampon instructions. Considérons les deux types de tampons mis en oeuvre, tampons à 2 et 4 lignes (figure 5). L'UC veut charger une instruction I (L_I est la ligne à laquelle appartient I). Alors:

1. l'antémémoire et le tampon sont lus simultanément si le bus antémémoire/UC n'est pas occupé par des mouvements de données.
2. seul le tampon est lu dans le cas contraire. Ensuite s'il y a un défaut de tampon, on lit l'antémémoire unifiée.

Tampon à 2 lignes:

- si $I \notin$ tampon:

  $I \notin$ antémémoire: il y a un défaut d'antémémoire. I est lue en mémoire principale.

  $I \in$ antémémoire: I est alors envoyée à l'UC et copiée dans le tampon. En fait c'est L_I qui est copiée dans le tampon.

- si $I \in$ tampon:

  $I \in$ *ou* $\notin$ antémémoire: I est envoyée du tampon à l'UC et simultanément la ligne suivant L_I (en termes d'adresse) est envoyée dans le tampon si elle appartient à l'antémémoire et si le bus antémémoire/UC n'est pas occupé par des mouvements de données.

Tampon à 4 lignes:

Le mécanisme est similaire au tampon à 2 lignes, mais en plus on fait un test sur le type des instructions.

- I n'est pas un saut:

  La stratégie est la même que pour le tampon à 2 lignes.

- I est un saut:

  Le but est de préserver la ligne à laquelle l'instruction de saut I appartient et la ligne cible de I.

  Les programmes étant en général composés de boucles, il est fréquent qu'un même saut soit effectué plusieurs fois. C'est pourquoi, une instruction de saut et sa cible sont systématiquement gardées dans deux des lignes du tampon (réservées pour ce type d'instructions) et les deux autres lignes sont consacrées à la ligne courante de l'instructions et à la ligne suivante (idem tampon à deux lignes).

Ce tampon instructions permet d'éviter le phénomène de rejets prématurés des lignes instructions, phénomème courant dans les antémémoires unifiées où les lignes instructions et données interfèrent.

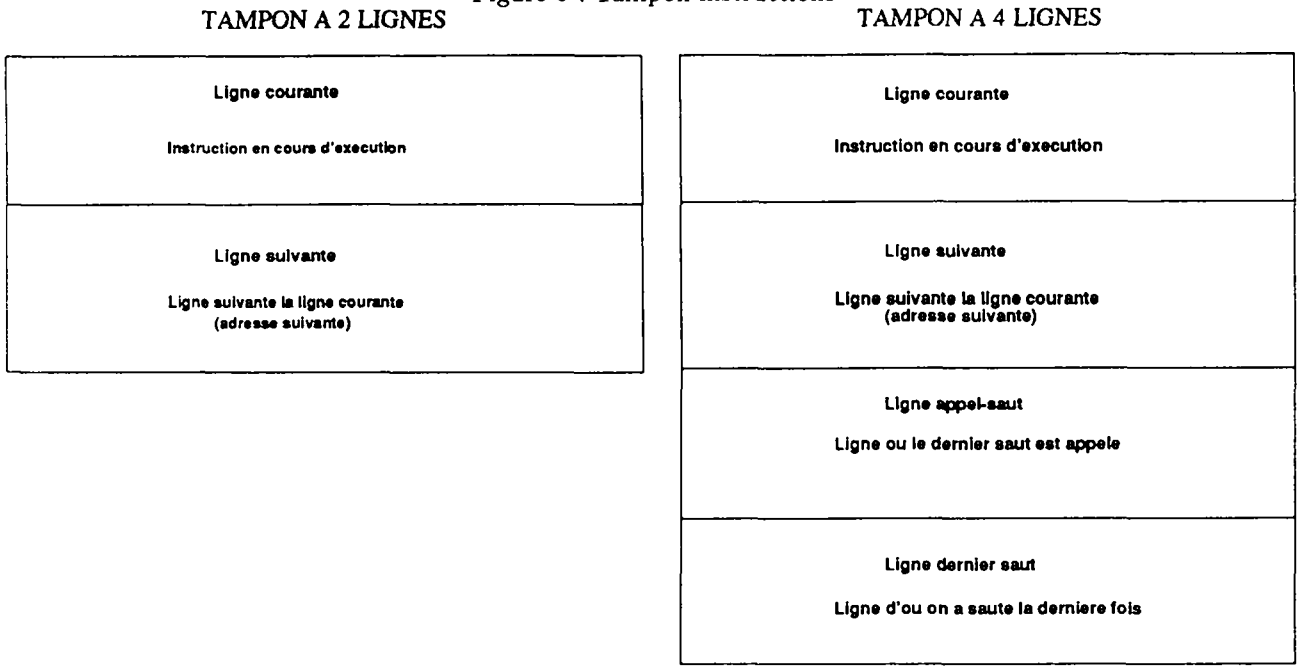
3.2 Antémémoires mixtes

Cette organisation d'antémémoires (figure 6) est composée de deux antémémoires: l'antémémoire dite *séparée* qui est destinée exclusivement aux instructions et l'antémémoire dite *unifiée* qui peut recevoir des données et des instructions. Ces deux antémémoires sont séparées physiquement et reliées par un bus unidirectionnel allant de l'antémémoire *unifiée* à l'antémémoire *séparée*.

3.2.1 Principe de l'antémémoire *unifiée* et de l'antémémoire *séparée*

Ce principe est basé sur la très grande localité spatiale des instructions. Lorsqu'une ligne instructions L_I est lue en mémoire principale, elle est automatiquement copiée dans l'antémémoire *séparée* et dans l'antémémoire *unifiée*. Ainsi il est possible de réduire la taille de l'antémémoire *séparée* sans affecter le nombre de miss instructions. En effet grâce à la copie simultanée de L_I dans les deux antémémoires, on a palié aux effets de rejets prématurés des lignes instructions dus à un espace de stockage trop réduit: les instructions récemment utilisées

Figure 5 : Tampon instructions



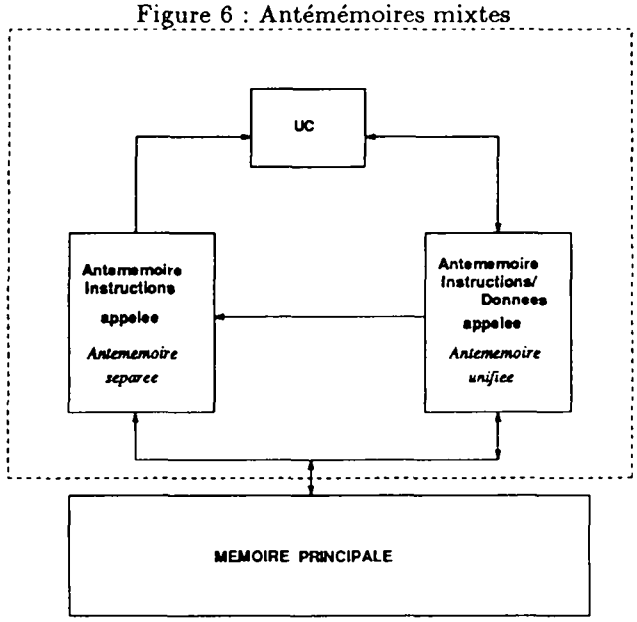
qui ont été rejetées de l'antémémoire instructions (antémémoire *séparée*) ont de fortes chances d'appartenir toujours à l'antémémoire *unifiée*.

3.2.2 Mécanisme de recherche et de remplacement dans les antémémoires mixtes

Suivant le type (donnée ou instruction) de la requête de l'UC, nous avons deux fonctionnements différents:

- requête d'une donnée: lorsque l'UC envoie une requête de type donnée, on regarde si elle appartient à l'antémémoire *unifiée*. Cette recherche peut être limitée aux lignes de données. Si la donnée n'appartient pas à l'antémémoire *unifiée*, on a un miss. Ce miss donnée est traité comme pour les antémémoires séparées avec une stratégie de remplacement LRU appliquée à toute l'antémémoire *unifiée* sans distinction de types.
- requête d'une instruction: soit I l'instruction à exécuter et L_I la ligne à laquelle elle appartient. Dans un premier temps, on regarde si L_I appartient à l'antémémoire *séparée*. Si c'est le cas la requête est satisfaite et on peut passer à la suivante, sinon on cherche I dans l'antémémoire *unifiée*:
  - si I appartient à l'antémémoire *unifiée*: on envoie I à l'UC et on copie L_I dans l'antémémoire *séparée* avec la stratégie LRU. L_I appartient toujours à l'antémémoire *unifiée*. La copie de L_I de l'antémémoire *unifiée* dans l'antémémoire *séparée* est considérée comme une référence à cette ligne. En effet si on applique la stratégie LRU à l'antémémoire *unifiée*, L_I sera considérée comme récemment référencée.
  - si I n'appartient pas à l'antémémoire *unifiée*: on est dans le cas d'un miss instructions. L_I est lue en mémoire principale, puis copiée dans l'antémémoire *séparée* et également dans l'antémémoire *unifiée*.

Figure 6 : Antémémoires mixtes



Nous verrons dans la section 4.3.1 (taille des antémémoires), l'importance de la répartition de l'espace mémoire entre l'antémémoire *séparée* et l'antémémoire *unifiée*. Ce phénomène fera l'objet d'une étude expérimentale.

3.3 Antémémoires semi-unifiées

Cette organisation est faite de deux antémémoires. Ces deux antémémoires sont séparées physiquement, mais peuvent recevoir toutes les deux des données et des instructions (figure 7).

3.3.1 Principe d'antémémoire principale-antémémoire secondaire

Pour les données (resp. les instructions), l'antémémoire dite de données (resp. d'instructions) est l'antémémoire principale et l'autre est l'antémémoire secondaire (figure 8).

- antémémoire principale: elle se comporte comme une antémémoire dédiée aux données (resp. instructions).
- antémémoire secondaire: elle reçoit les données (resp. instructions) qui ont été la cible de remplacements dans l'antémémoire principale. Ces données (resp. instructions) transitent par cette antémémoire. Elle se comporte comme un niveau mémoire supplémentaire entre l'antémémoire principale et la mémoire principale.

Ce principe permet d'augmenter en fonction des besoins l'espace de stockage d'un type (instruction ou donnée) d'éléments donné, mais de gérer ce besoin afin qu'il ne perturbe pas l'espace indispensable à l'autre type (donnée ou instruction). De plus l'utilisation de deux antémémoires séparées physiquement permet à l'UC de lire simultanément des instructions et des données.

Figure 7 : Antémémoires semi-unifiées

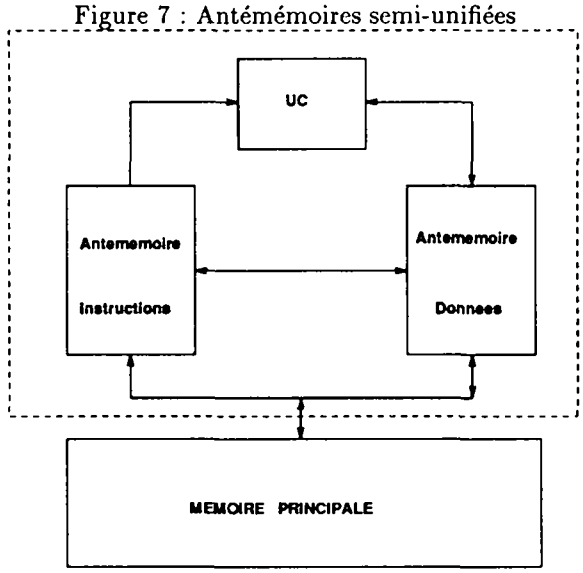
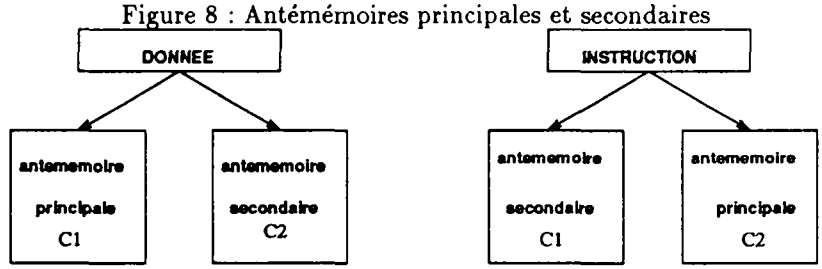


Figure 8 : Antémémoires principales et secondaires



3.3.2 Stratégie de remplacement pour antémémoires semi-unifiées

La ligne L_I de l'instruction à exécuter I n'est pas dans l'antémémoire principale (C^1), alors:

- si I appartient à l'antémémoire secondaire (C^2):

  I est lue dans cette antémémoire C^2 et envoyée à l'UC. L_I est permutée avec une ligne choisie dans l'antémémoire principale C^1. Cette ligne de permutation sera choisie avec la stratégie LRU. Nous proposons plus loin une gestion architecturale de cette permutation.

- si I n'appartient pas à l'antémémoire secondaire (C^2):

  I est lue alors en mémoire principale et copiée dans l'antémémoire principale (C^1). En posant $\mathcal{E}^1 = \mathcal{E}^1_{don} \bigcup \mathcal{E}^1_{inst}$ où $\mathcal{E}^1_{don}$ est l'ensemble des lignes de C^1 contenant des données et $\mathcal{E}^1_{inst}$ est l'ensemble des lignes de C^1 contenant des instructions susceptibles de recevoir L_I. La ligne cible du remplacement L_{cible} est choisie comme suit.

  - si $card(\mathcal{E}^1_{don}) \neq 0$

    La ligne cible L_{cible} est choisie dans $\mathcal{E}^1_{don}$ avec la stratégie LRU.

- si $card(\mathcal{E}^1_{don}) = 0$

  La ligne cible L_{cible} est choisie avec la stratégie LRU. Cette ligne est soit écrasée, soit recopiée dans l'antémémoire secondaire C^2. Elle est recopiée dans l'antémémoire secondaire avec la stratégie OLI.

  La stratégie OLI (Old Lines Instruction) est telle que:

  - Soit $\mathcal{E}^2(L_{cible})$ contient des vieilles lignes ($V^2(L_{cible}) \neq 0$). Alors une nouvelle ligne cible est choisie pour L_{cible} suivant LRU dans $V^2(L_{cible})$.
  - Soit $\mathcal{E}^2(L_{cible})$ ne contient pas de vieilles lignes ($V^2(L_{cible}) = 0$). Alors:
    - $\mathcal{E}^2_{inst} \neq 0$: la nouvelle ligne cible est choisie suivant LRU dans $\mathcal{E}^2_{inst}$
    - $\mathcal{E}^2_{inst} = 0$: la ligne L_{cible} n'est pas conservée.

Remarques:

1. La stratégie de remplacement s'applique de la même manière pour les données (remplacer *instructions* par *données* et C^1 par C^2).

2. Gestion architecturale de la permutation:

   Il s'agit de la permutation entre une ligne de l'antémémoire principale et une ligne de l'antémémoire secondaire. Afin de permuter ces deux lignes, nous proposons d'utiliser le victim cache ([JOUPPI90]). Ce victim cache est un tampon auquel est rajouté une ligne qui servira à stocker la ligne de permutation issue de C^1. La ligne issue de C^2 est transférée via le bus placé entre C^2 et C^1 simultanément à la copie de la ligne issue de C^1 dans le victim cache.

4 Performances

Nous allons dans la section 4.1 décrire le simulateur mis en oeuvre et discuter sur les problèmes que posent la comparaison d'architectures hétéroclites. Ensuite dans la section 4.2 nous effectuerons l'étude théorique du temps d'exécution en présentant au préalable le modèle choisi. Enfin dans la section 4.3 nous étudierons les performances des différentes organisations d'antémémoires en fonction de leurs paramètres physiques et de la latence de la mémoire principale. La description des traces utilisées est faite en annexe (A.2).

4.1 Simulation

La validation des architectures actuelles s'effectue de manière empirique, expérimentation et simulation [PAT90]. Pour nos études, nous avons donc mis en oeuvre un simulateur d'antémémoires reconfigurable en C++ [MET90]. Ce simulateur est reconfigurable en fonction:

- des architectures d'antémémoires décrites précédemment
- des politiques de remplacement dans les antémémoires
- de la taille des antémémoires et des tampons
- de la taille de la ligne
- du degré d'associativité des antémémoires

- de la latence et de la taille de la mémoire principale

Il faut également noter que pour les différentes simulations, nous utilisons des **victim caches** [JOUPPI90] afin de réduire les conflits lors de miss. Par ailleurs les simulations s'effectuent à partir de traces correspondant à des portions de l'exécution de différents programmes.

Pour chaque organisation d'antémémoires, nous avons simulé différentes valeurs des paramètres:

- taille des antémémoires $\in$ [1 Ko, 32 Ko]
- taille de la ligne $\in$ [16 o, 264 o]
- degré d'associativité $\in$ { 1, 2 , 4 }
- stratégie de remplacement: Random, LRU et stratégies proposées dans la description de certaines organisations d'antémémoires.
- politique d'écriture: recopie

Notations utilisées pour les différentes organisations d'antémémoires

- SEP: correspond aux antémémoires séparées
- UNI: correspond à l'antémémoire unifiée avec tampon instructions
- MIX: correspond aux antémémoires mixtes
- SU: correspond aux antémémoires semi-unifiées
- E_{ant}: correspond à l'espace de stockage sur la puce du processeur. $E_{ant} = E_{don} \bigcup E_{inst}$ où E_{don} est l'espace de stockage pouvant recevoir des données et E_{inst}, l'espace de stockage pouvant recevoir des instructions.

Remarques:

1. Les différentes tailles d'antémémoires et de lignes sont données en octets (1 octet = 1 o et 1 Ko = 1024 o). En général ces tailles sont des puissances de 2.

2. Les temps d'exécutions et les latences sont donnés en nombre de cycles.

Cohérences des paramètres

Afin de comparer les différentes architectures d'antémémoires, les valeurs des paramètres physiques doivent être cohérentes.

- taille des antémémoires: deux antémémoires séparées de taille $T(C^i)$ avec $i \in \{1,2\}$ sont en correspondance avec une antémémoire unifiée de taille $T(C)$: $T(C^1) + T(C^2) = T(C)$.
- taille de la ligne et latence mémoire: elles sont les mêmes pour deux architectures distinctes.
- degré d'associativité A: il faut respecter l'homogénéité des degrés (exemple: $A(C^1_{separe})$ $= A(C^2_{separe}) = A$ alors $A(C^1_{semi-unifie}) = A(C^2_{semi-unifie}) = A$).

Pour mener une étude rigoureuse, il nous faut simuler et analyser le modèle utilisé.

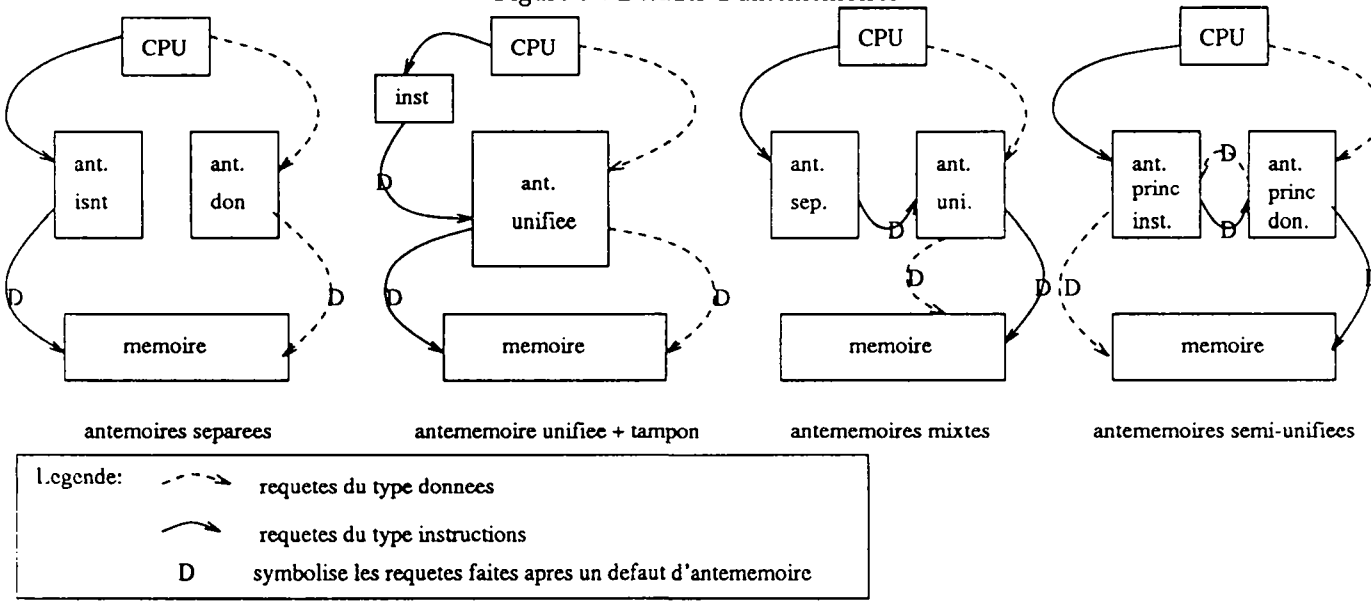
4.2 Etude théorique

Certains paramètres du temps d'exécution T_{exec} ne dépendent pas de la hiérarchie mémoire (cf. 2.1.3), ils peuvent être fixés selon les valeurs en usage dans les architectures réelles. Nous présentons un modèle simple pour le CPU et la mémoire. Biensur nous sommes toujours dans le cadre des architectures RISC avec technique du pipeline (figure 10).

4.2.1 Modèle

- On considère que toutes instructions et données lues dans une des antémémoires à moins qu'elles n'engendrent un défaut d'antémémoires ne coûte qu'un cycle. Les défauts considérés sont visualisés sur la figure 9.

Figure 9 : Défauts d'antémémoires



- Le premier mot d'une ligne lue en mémoire principale est le mot demandé par l'UC: $T_2 = T_{lat}$.
- Excepté pour l'antémémoire unifiée sans tampon instructions, le transfert des données et des instructions se fait en parallèle.

4.2.2 Approximation et étude du temps d'exécution

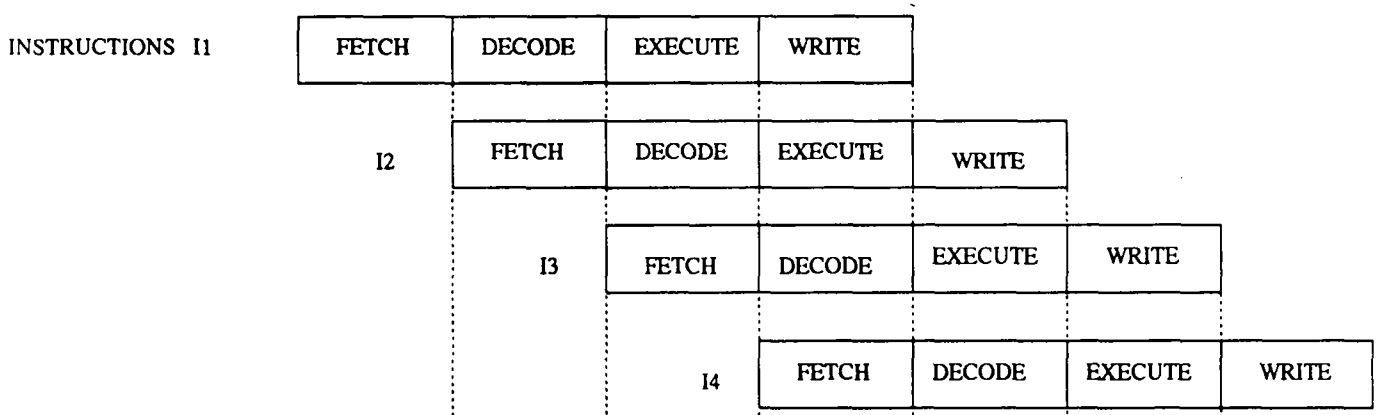
Pour chaque organisation, nous allons donner une approximation du temps d'exécution T_{exec} compte tenu du modèle utilisé.

Figure 10 : Pipeline instructions

Suite d'instructions simulees:

I1 : LOAD R1, A
I2 : LOAD R2, B
I3 : ADD R3, R1, R2
I4 : STORE C, R3

Pipeline instructions:



Occupations des bus:

Temps	t1	t2	t3	t4	t5	t6	t7
Bus instruction	I1	I2	I3	I4			
Bus donnees			A	B		C	

Antémémoires séparées:

$$T_{exec} = N_{inst} + N_{miss} * T_{lat} \tag{5}$$

- N_{miss}: nombre de miss sur les antémémoires séparées: $N_{miss} = (N_{inst} + N_{don}) * \lambda_{miss}$.
- T_{lat}: latence de la mémoire principale.

Antémémoire unifiée sans tampon instructions:

$$T_{exec} = N_{inst} + (N_{inst} + N_{don}) * (\lambda_{hit}^{donnees} * T_1 + \lambda_{miss} * T_{lat}) \tag{6}$$

- dans l'organisation architecturale d'antémémoire unifiée sans tampon, toutes les références (données et instructions) demandent au moins un cycle pour être transferées dans l'UC. Il est donc clair que

ses performances seront nettement inférieures à celles des autres architectures. C'est pourquoi nous ne ferons pas d'étude approfondie de cette organisation.

Antémémoire unifiée avec tampon instructions:

$$T_{exec} = N_{inst} + (N^u_{miss} * T_{lat}) + (N^b_{miss} * T^b_{lat}) \quad (7)$$

- N^u_{miss}: nombre de miss sur l'antémémoire unifiée.
- T_{lat}: latence de la mémoire principale.
- N^b_{miss}: nombre de miss sur le tampon instructions. Nous avons expliciter le mécanisme de remplissage du tampon instructions dans la section 3.1. Il y a un miss sur le tampon lorsque l'on veut charger une instruction I dans l'UC et que cette instruction I n'appartient pas au tampon et que le bus antémémoire/UC est occupé par des mouvements de données.
- T^b_{lat}: nombre de cycles pour lire une instruction dans l'antémémoire unifiée lorsque l'on a un défaut de tampon.

Antémémoires mixtes:

$$T_{exec} = N_{inst} + (N^s_{miss} * T^s_{lat}) + (N^u_{miss} * T_{lat}) \quad (8)$$

- N^s_{miss}: nombre de miss instructions sur l'antémémoire *séparée*.
- T^s_{lat}: nombres de cycles pour lire une instruction dans l'antémémoire *unifiée* lorsqu'elle n'appartient pas à l'antémémoire *séparée*.
- N^u_{miss}: nombre de miss sur l'antémémoire *unifiée*. Ce nombre est obtenu en additionnant tous les miss données ainsi que les miss instructions (données et instructions n'appartiennant ni à l'antémémoire *séparée*, ni à l'antémémoire *unifiée*).
- T_{lat}: latence de la mémoire principale.

Antémémoires semi-unifiées:

$$T_{exec} = N_{inst} + (N^p_{miss} * T^p_{lat}) + (N^s_{miss} * T_{lat}) \quad (9)$$

- N^p_{miss}: nombre de miss sur l'antémémoire principale. Ce nombre est obtenu en additionnant les miss instructions sur C^1 et les miss données sur C^2 (figure 7).
- T^p_{lat}: nombre de cycles pour lire un élément dans l'antémémoire secondaire s'il n'appartient pas à l'antémémoire principale.
- N^s_{miss}: nombre de miss dans l'antémémoire secondaire (miss instructions sur C^2 et miss données sur C^1).
- T_{lat}: latence de la mémoire principale.

- **Etude de T_{exec}**

  Nous voulons montrer que les organisations d'antémémoire unifiée avec tampon instructions, d'antémémoires mixtes et d'antémémoires semi-unifiées donnent en général de meilleures performances que l'organisation d'antémémoires séparées.

  Certains paramètres de T_{exec} n'ont pas d'expression littéral (N_{miss}, N^u_{miss}, N^b_{miss}, N^p_{miss}, N^s_{miss}) aussi la preuve théorique à l'aide des formules (5), (7), (8), (9) est très difficile. Cependant nous allons mener une étude à partir de la simulation de programmes qui permet d'obtenir les paramètres N_{inst}, N_{miss}, N^u_{miss}, N^b_{miss}, N^p_{miss} et N^s_{miss}.

  - Analyse des courbes (figure 11):

    Nous présentons figure 11 les performances obtenues en moyennant les temps d'exécution et les nombres d'instructions des différents programmes simulés; ces performances sont fonctions du coût des échanges entre les antémémoires (tableau 2) et de la latence mémoire. Les résultats présentés sont caractéristiques du MIPS R4000 où *lat* (temps d'échange entre antémémoires) compte tenu de la structure du pipeline devrait être égal à 3 cycles. La droite c1 représente les performances du MIPS R4000 avec un seul niveau d'antémémoire (latence mémoire approximativement égale à 25 cycles) et la droite c2 représente les performances du MIPS R4000 avec une antémémoire secondaire (latence mémoire approximativement égale à 8 cycles). Ainsi les organisations UNI, MIX et SU (cf notations 4.1) donnent de meilleures performances que l'organisation SEP dès que la latence mémoire dépasse un certain seuil, seuil qui dépend des latences de défaut de tampon, d'antémémoire *séparée* ou d'antémémoire principale propres à chaque organisation (*lat*). Ce seuil peut varier de 2-3 cycles pour les antémémoires semi-unifiées 1-way avec $lat = 1$ cycle à 35 cycles pour les antémémoires unifiées 4-way avec $lat = 3$ cycles.

    Il est alors tout à fait envisageable de mettre en oeuvre ces organisations dans les cas des microprocesseurs.

Table 2 : Transferts entre antémémoires

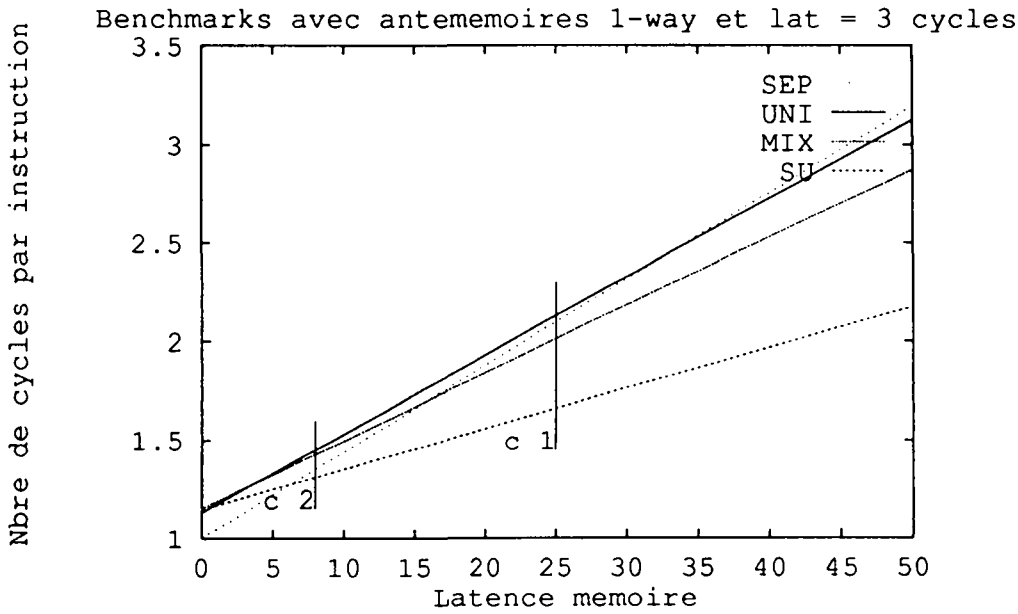
Organisation d'antémémoires	Transfert de A	vers B	Coût du transfert *lat* (figure 11)
UNI	antémémoire unifiée	tampon instructions	T^b_{lat}
MIX	antémémoire *unifiée*	antémémoire *séparée*	T^s_{lat}
SU	antémémoire secondaire antémémoire principale	antémémoire principale antémémoire secondaire	T^p_{lat}

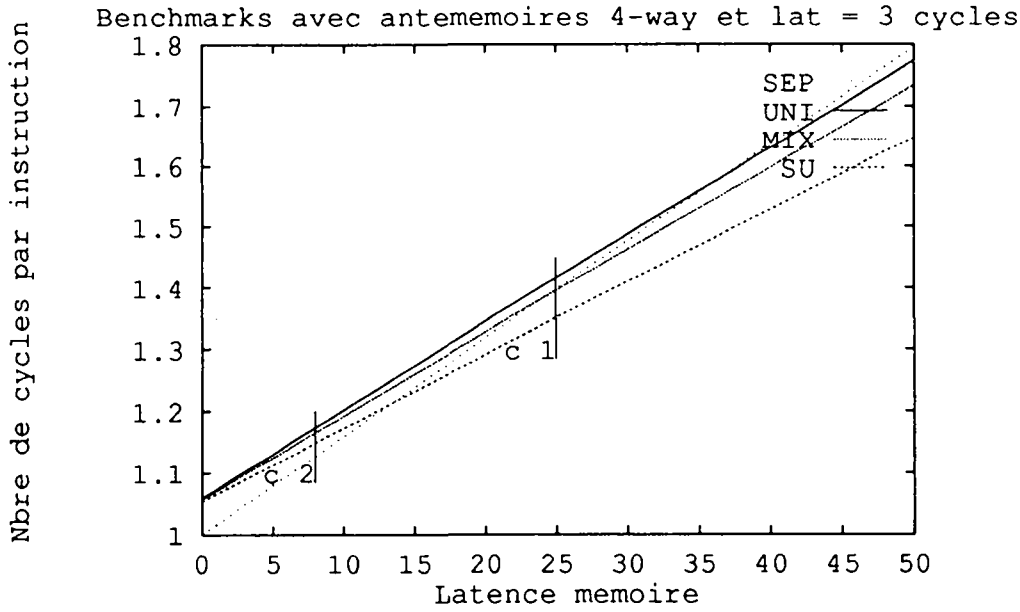
Mais nous reviendrons plus en détail sur cette comparaison entre les organisations architecturales d'antémémoires dans notre étude de performances (4.3).

4.3 Etude des performances

Les différents paramètres des antémémoires (taille des antémémoires, taille de la ligne, degré d'associativité) ont fait l'objet de nombreuses études qui ont prouvé que les performances d'une organisation d'antémémoires

Figure 11 : Etude des performances en fonction de T_{lat}, T^b_{lat}, T^p_{lat}. Paramètres fixés: $E_{ant} = 8$ Ko et $L = 64$ o.





étaient sensibles aux variations de ces paramètres. Pour notre étude, les paramètres physiques des organisations architecturales d'antémémoires seront étudiés séparément. A l'exception du paramètre considéré, les paramètres sont fixés aux valeurs optimales, déterminées empiriquement.

4.3.1 Paramètres physiques des antémémoires

1. **Taille des antémémoires**

   La taille des antémémoires [SMITH82] est liée à la place laissée sur la puce du processeur à des fins de stockage (E_{ant}). Cette place, compte tenue de la technologie actuelle n'est pas très importante. La capacité de stockage pour les processeurs intégrant des antémémoires sur leur puce est comprise entre 12 et 36 Ko, ceci pour les processeurs référencés dans le tableau 1.

   - Partition de l'espace de stockage sur la puce du processeur:

     Nous avons constaté que pour obtenir des résultats optimaux sur la plus grande partie de nos benchmarks, les tailles des antémémoires séparées doivent être égales. Ce phénomène est également valable pour les antémémoires semi-unifiées. Par contre en ce qui concerne l'organisation des antémémoires mixtes, la répartion de l'espace mémoire entre les deux antémémoires joue un rôle prépondérant.

     Nous présentons dans le tableau 3, les performances obtenues pour le benchmark cptc (taille de ligne = 64 o et associativité = 2) en fonction de la partition entre l'antémémoire données et l'antémémoire instructions pour SEP et entre l'antémémoire *séparée* et l'antémémoire *unifiée* pour MIX. Ces résultats montrent que MIX peut donner de meilleures performances s'il y a un déséquilibre entre les tailles des deux antémémoires (la taille de l'antémémoire *unifiée* plus grande que la taille de l'antémémoire *séparée*). On constate également que pour une augmentation relativement faible de l'espace de stockage, la partition: antémémoire *unifiée* = 8 Ko et antémémoire *séparée* = 512 o, donne de meilleurs résultats que toutes les partitions obtenues avec un espace de stockage de 8 Ko (il en est de même avec un espace de 16 Ko où la partition antémémoire *unifiée* = 16 Ko et antémémoire *séparée* = 512 o est optimale). Par conséquent dans nos différentes études, nous partitionnerons l'espace de stockage sur la puce du processeur en prenant une grande antémémoire *unifiée* et une petite antémémoire *séparée* quitte à augmenter légerement la taille de cette espace.

   - Analyse des courbes (figures 12 et 13):
     - les performances sont meilleures pour UNI et SU lorsque l'on a de petites tailles d'antémémoires ($E_{ant} < 8$ Ko). Ce phénomène est lié à la non différenciation entre les données et les instructions dans UNI et SU. En effet les deux types d'éléments bénéficient en réalité d'un espace de stockage bien supérieur à la moitié de l'espace. On a les même résultats pour MIX mais avec un certain décalage. En effet si on prend un espace de stockage de 8 Ko pour SEP, il faut considérer un espace de 8 Ko + 512 o pour MIX afin d'obtenir de meilleures performances.
     - dans toutes les simulations, on constate qu'il existe une taille d'antémémoire limite au dessus de laquelle les performances ne sont guère améliorées ($E_{ant} > 8$ Ko (à 512 o près) dans la figure 12 et $E_{ant} > 16$ Ko (à 512 o près) dans la figure 13).

2. **Taille de la ligne**

   La taille de la ligne [SMITH87] est souvent fonction de la largeur du bus entre l'antémémoire et la mémoire principale. Le choix de cette taille est lié au temps de chargement d'une ligne de la mémoire principale vers l'antémémoire (appartenant à la puce du CPU). Elle dépend donc de la latence de la mémoire principale.

   Nous présentons dans le tableau 4 certains avantages et inconvénients liés à la taille de la ligne (grande ou petite).

Table 3 : Partition de l'espace mémoire E_{ant} pour SEP et MIX

Nombre de miss(*) en fonction de la partition de l'espace de stockage pour l'organisation d'antémémoires séparées SEP

Taille antémémoire données (o)	512	1024	2048	4096	8192	16384
Taille antémémoires instructions (o) 512	100	89.7	80.4	75.3	12.3	71.2
1024	76.7	66.4	57.2	52	49.1	-
2048	60.9	50.7	41.4	36.3	33.4	-
4096	52.2	38.7	29.4	24.3	21.4	-
8192	41.8	31.6	22.3	17.2	14.3	-

Nombre de miss(*) en fonction de la partition de l'espace de stockage pour l'organisation d'antémémoires mixtes MIX

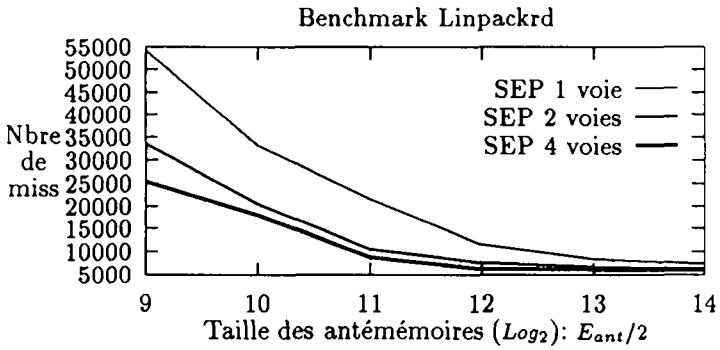
Taille antémémoire *unifiée* (o)	512	1024	2048	4096	8192	16384
Taille antémémoire *séparée* (o) 512	100	82.2	52.5	34	20.2	11.1
1024	74.8	64.1	46.7	33.1	20.1	-
2048	58.8	49.7	41.4	31.8	20	-
4096	46.5	37.5	30.1	-24.4-	17.9	-
8192	39.1	30	22.6	17.4	13.7	-

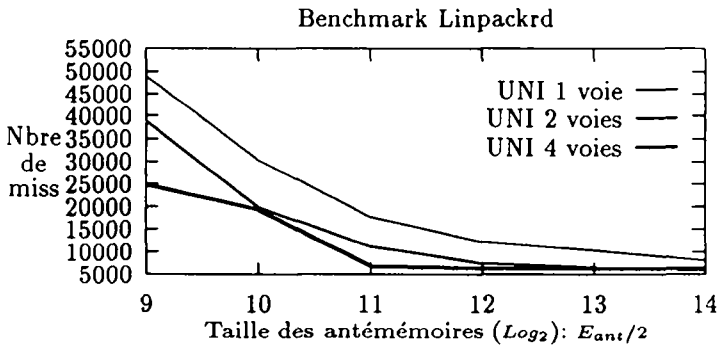
(*): la répartition pour laquelle on obtient le plus grand nombre de miss est prise comme référence (répartition avec deux antémémoires de 512 o). Exemple: *nombre de miss pour MIX telle que (taille antémémoire séparée = 4096 o, taille antémémoire unifiée = 4096 o)* = 24.4 % * *nombre de miss pour MIX telle que (taille antémémoire séparée = 512 o, taille antémémoire unifiée = 512 o).*

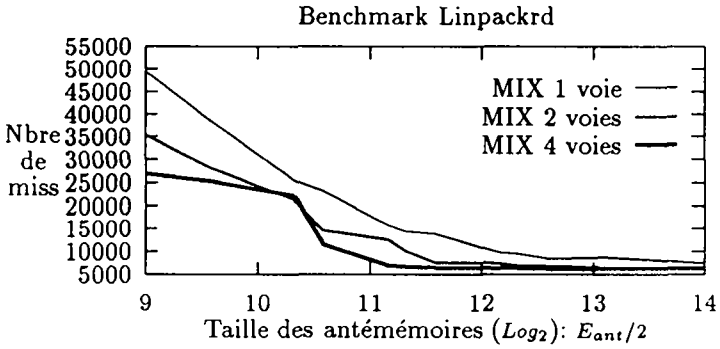
Table 4 : Quelle taille de ligne choisir?

	GRANDE LIGNE	PETITE LIGNE
AVANTAGES	- utile pour les gros ordinateurs où la bande passante de la mémoire principale est large - est meilleure quand la latence domine sur les transferts	- réduit le nombre de traffics mémoire - minimise la compléxité de la logique - est meilleure quand les transferts dominent sur la latence
INCONVENIENTS	- génère des transferts longs - augmente le traffic et les interferences - augmente le traffic du aux mises à jour de la mémoire (WB ou WT) - n'utilise plus les propriétés de localité temporelle	- donne une latence mémoire élevée par rapport au temps de chargement de la ligne - demande une place importante dans l'antémémoire pour le stockage des tags proportionnelement aux données

Figure 12 : Influence de la taille des antémémoires sur le nombre de miss. Paramètre fixé: $L = 64$ o.







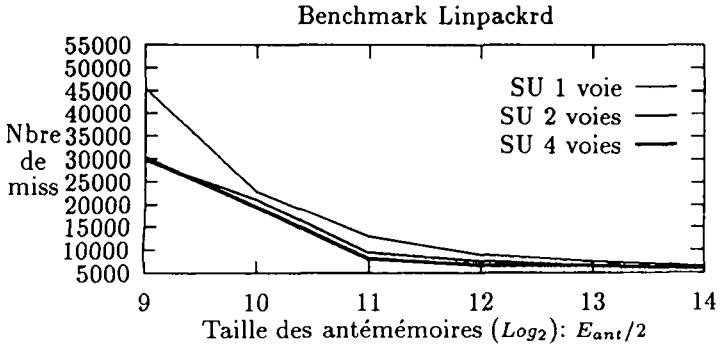
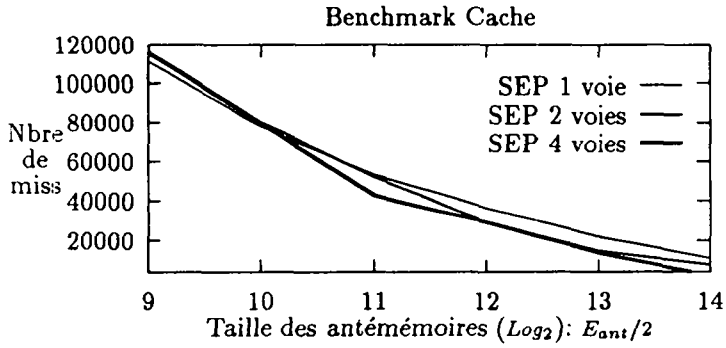
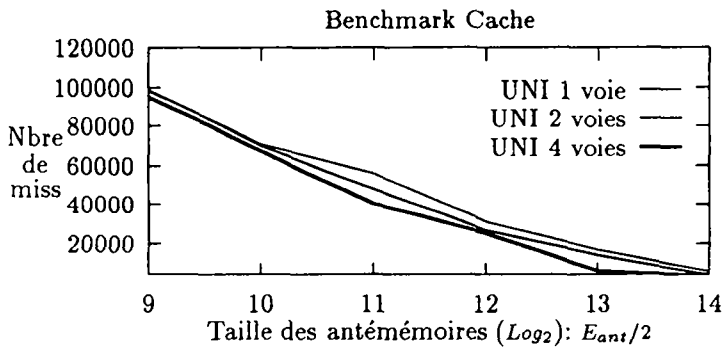
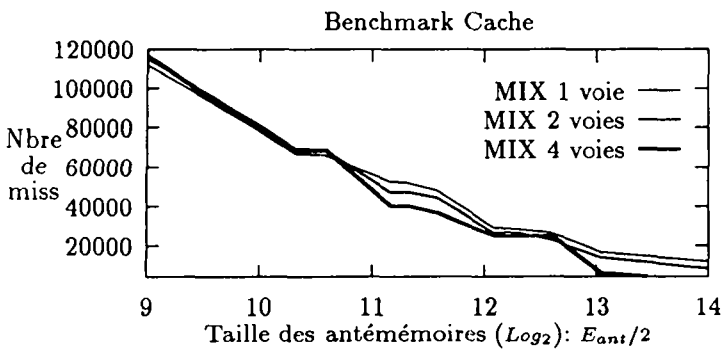
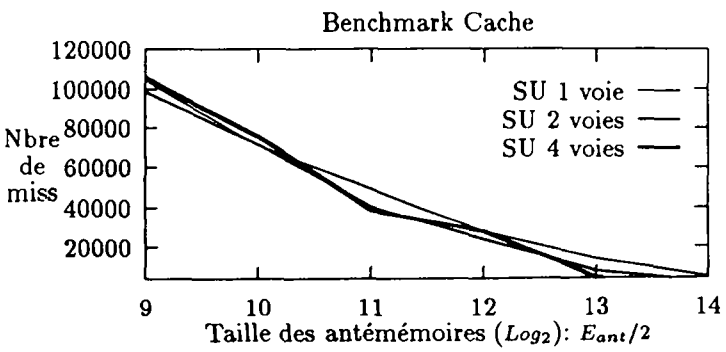


Figure 13 : Influence de la taille des antémémoires sur le nombre de miss. Paramètre fixé: $L = 64$ o.









- Analyse des courbes (figures 14 et 15):
  - pour des tailles de lignes petites ($L < 64$ o), UNI, MIX et SU donnent de meilleurs résultats. En effet ces organisations permettent de conserver un plus grand nombre de lignes instructions ou données, autorisant une meilleure exploitation de la localité temporelle.
  - par contre il existe un seuil ($L \geq 256$ o) au delà duquel les performances de UNI, MIX et SU fléchissent. En effet, les antémémoires unifiées, mixtes et semi-unifiées autorisent plus de souplesse, mais sont plus sensibles aux phénomènes de pollution. Or ceux-ci sont aggravés par des grandes tailles de lignes.
  - la taille de lignes optimale semble se situer entre 64 o et 128 o.

3. Degré d'associativité

Le choix du degré d'associativité [HILL89] est lié en grande partie à un problème de coût. En effet l'augmentation de l'associativité des antémémoires augmente leurs coûts. Une conséquence est que seules les petites mémoires peuvent être totalement associatives. Par contre le temps d'accès n'augmente plus en fonction du degré d'associativité car les recherches de références dans les différents bancs de l'antémémoire se font en parallèle. A noter que les antémémoires *direct-mapped* permettent d'utiliser un cycle plus court (exécution optimiste) [HILL88] et sont, de ce fait, préférées par certains constructeurs.

Si nous nous référons aux études de [HILL89] menées sur un important échantillon d'antémémoires (taille des antémémoires, taille des lignes, degré d'associativité variables) et de programmes, les améliorations de performances par rapport au degré d'associativité sont telles que:

n-ways	1 à 2	2 à 4	4 à 8
Améliorations en %	30	10	5

- Analyse des courbes (figures 12, 13, 14, 15):
  - pour certains benchmarks (12), (14), (15), l'amélioration des performances pour SEP, UNI et MIX entre les différents degrés d'associativité A est considérable notamment entre $A = 1$ et $A = 2$. Tandis que pour SU les améliorations sont beaucoup moins significatives. En fait SU a quelques similarités avec le principe d'associativité. Virtuellement une organisation SU avec $A = 1$ se comporte un peu comme une organisation UNI ou SEP avec $A = 2$.
  - pour d'autres benchmarks (figure (13)), lorsque l'on a une petite taille d'antémémoire le degré d'associativité A est très peu influant.
  - si on considère une taille d'antémémoire maximisant le performances, l'augmentation du degré d'associativité A de 1 à 2 améliore les performances. Par contre entre $A = 2$ et $A = 4$, les améliorations sont faibles.

Récapitulation des résultats pour les différentes figures

Nous avons donc simulé différents programmes (voir A.2) en faisant varier les paramètres physiques des antémémoires. Nous présentons ici sous forme de tableau les résultats obtenus. Les graphes sont significatifs des différents comportements rencontrés. Le phénomène le plus remarquable est que **chaque organisation d'antémémoires** donne les **même paramètres optimaux**.

Paramètres	E_{ant}: espace de stockage	L: taille de la ligne	A: associativité
Paramètres fixés	- $L = 64$ o	- $E_{ant} = 8$ Ko	- $E_{ant} = 8$ Ko - $L = 64$
Figures	12, 13	14, 15	12, 13, 14, 15
Paramétres optimaux	- fig 12: $E_{ant} = 8$ Ko - fig 13: $E_{ant} = 16$ Ko	- fig 14: $L = 64$ o - fig 15: $L = 128$ o	- 1-way à 2 ways: bonnes améliorations - 2-ways à 4-ways: peu d'améliorations

4.3.2 Synthèse et comparaison

Nous avons mené toutes ces études afin de prouver que les organisations d'antémémoires proposées UNI, MIX et SU donnent de meilleurs résultats que SEP. Nous avons regroupé les résultats des simulations pour différents benchmarks [PNEV90]. Les graphes présentés figures (16), (17), (18) sont tels que:

- les benchmarks sont en abscisse (présentation en annexe (A.2)).
- en ordonnée, pour chaque organisation d'antémémoires, on donne la valeur du rapport: *nombre de miss pour une organisation d'antémémoires donnée* sur *nombre de miss pour la plus mauvaise organisation d'antémémoire*. En fait *cette plus mauvaise organisation* est *l'organisation SEP avec* $A = 1$.

Les améliorations de performances de SU par rapport à SEP sont très significatives pour un degré d'associativité de 1. Pour certains benchmarks, UNI n'améliore pas les performances par rapport à SEP. MIX est plus stable que UNI pour laquelle il y a parfois un déséquilibre involontaire entre données et instructions. Mais plus le degré d'associativité est grand, moins les différences en performances sont importantes.

SU est l'organisation qui donne les meilleures performances. Elle évite certains phénomènes parasites (ex: effet *ping-pong*) par rapport à UNI et offre de plus grandes possibilités dans le remplacement d'une ligne de l'antémémoire par rapport à MIX. D'autres figures de comparaison entre les différentes organisations architecturales d'antémémoires sont présentées en annexe B.

5 Conclusion

Le choix d'une organisation d'antémémoires influe sur le temps d'exécution des programmes. Notre étude a eu pour but de trouver une organisation d'antémémoires optimale respectant les caractéristiques des mémoires des microprocesseurs. Ces caractéristiques sont principalement liées à:

- la taille de stockage sur la puce du processeur qui est limitée
- la latence de la mémoire principale qui est assez élevée.

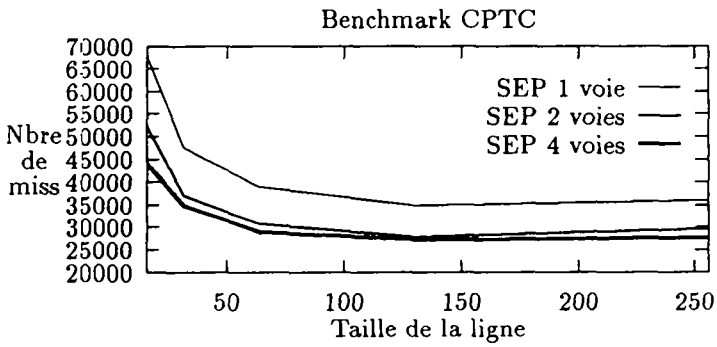
Nous avons proposé trois organisations d'antémémoires: l'antémémoire unifiée avec tampon instructions, les antémémoires mixtes et les antémémoires semi-unifiées. L'antémémoire unifiée avec tampon instructions a été conçue dans le but de pallier aux problèmes de séquencement non simultané des données et des instructions (problèmes rencontrés dans l'antémémoire unifiée habituellement utilisée), d'utiliser la localité spatiale des instructions avec le chargement à priori de certaines lignes d'instructions dans le tampon et de limiter les rejets prématurés de lignes instructions en cours d'exécution en les stockant dans ce même tampon. Mais cette organisation n'apporte pas toujours de nettes améliorations. En effet on assiste parfois à des conflits importants entre les lignes instructions et les lignes données.

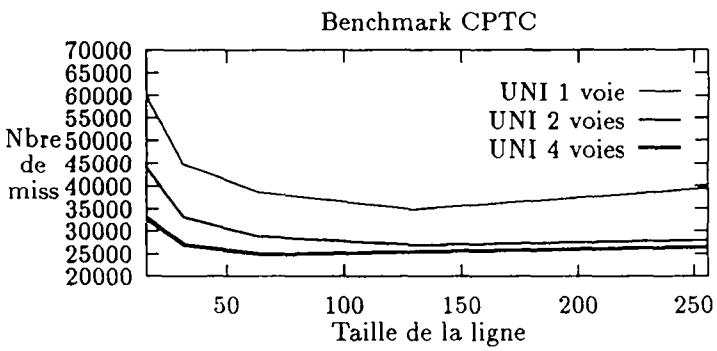
Les antémémoires mixtes et semi-unifiées ont elles été conçues afin de pallier aux carences des antémémoires séparées. Ces carences sont d'une part l'espace trop restreint dédié à chaque type d'éléments (données ou instructions) et d'autre part, le manque de souplesse et d'adaptabilité qui engendre un gaspillage de l'espace mémoire s'il y a un déséquilibre important entre les besoins en données et en instructions. En utilisant une petite

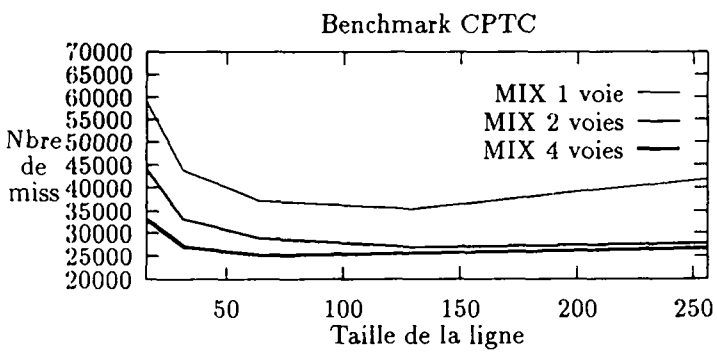
antémémoire instructions et une grande antémémoires unifiées (instructions et données), cas des antémémoires mixtes, ou en utilisant une des deux antémémoires comme antémémoire secondaire virtuelle pour l'autre, cas des antémémoires semi-unifiées, les éléments utiles sont éjectés moins rapidement et donc ont plus de chances d'être présents (au niveau des antémémoires) quand ils sont nécessaires. Par contre ce qui fait la puissance des antémémoires semi-unifiées, c'est l'augmentation artificielle du degré d'associativité associé à un type d'éléments. L'organisation architecturale des antémémoires semi-unifiées entraine une amélioration significative des performances et ceci par rapport aux organisations utilisées actuellement dans la plupart des microprocesseurs. Cette organisation parait particulièrement adaptée dans le cas où, pour des raisons de cycle machine, on choisit d'utiliser des antémémoires *direct-mapped*.

Ainsi en modifiant simplement quelques éléments architecturaux du processeur, les temps d'exécution des programmes peuvent être diminués. Un prolongement possible à cette étude serait d'introduire cette organisation d'antémémoires semi-unifiées dans un environnement multiprocesseurs, environnement où la diminution du nombre de communications avec la mémoire principale est primordiale.

Figure 14 : Influence de la taille de ligne pour chaque organisation d'antémémoires. Paramètre fixé: E_{ant} = 8 Ko.







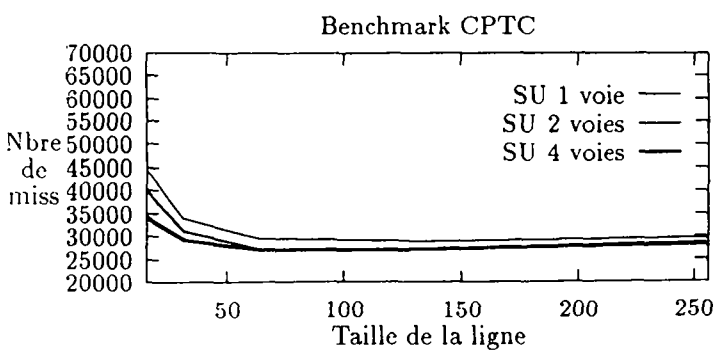
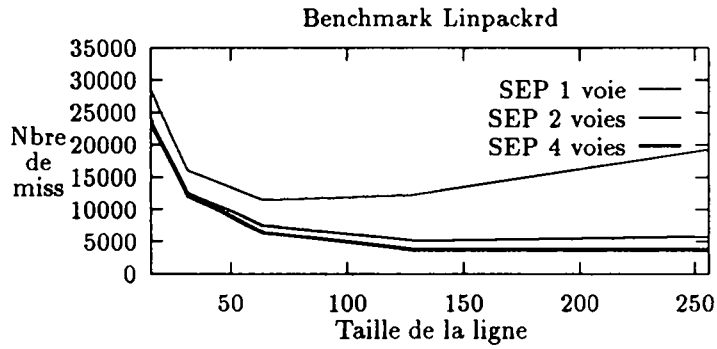
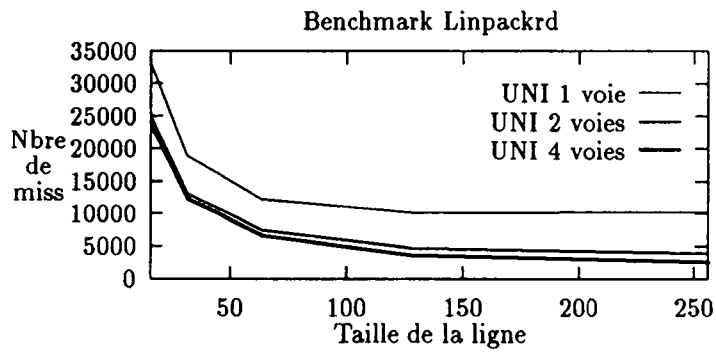
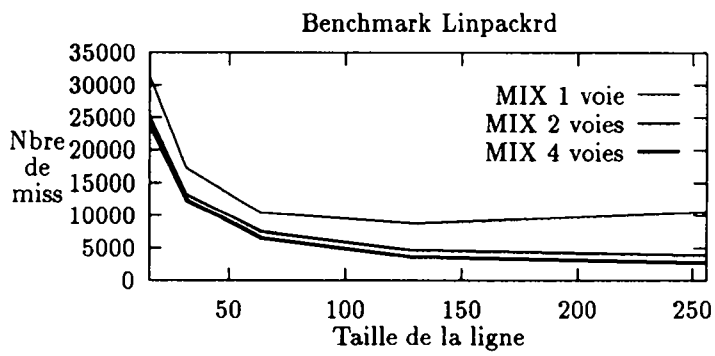


Figure 15 : Influence de la taille de ligne pour chaque organisation d'antémémoires. Paramètre fixé: E_{ant} = 8 Ko.







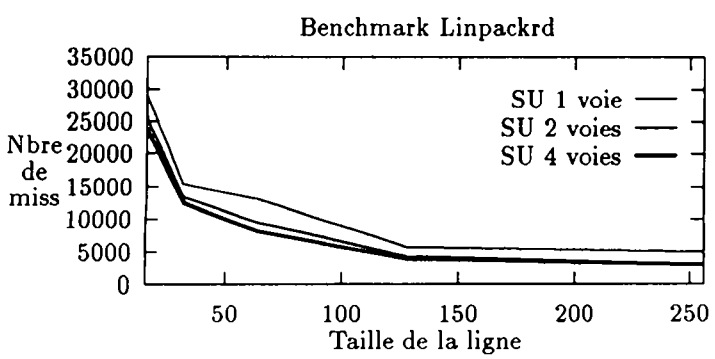


Figure 16: Comparaisons d'antememoires (8 Ko) 1-way

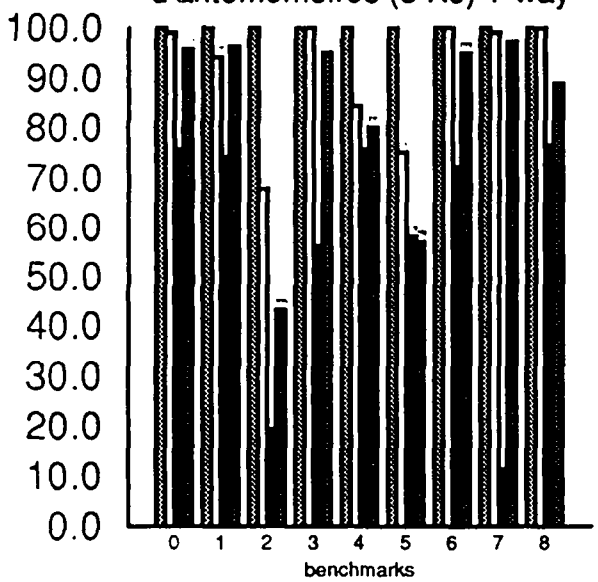


Figure 17: Comparaisons d'antememoires (8 Ko) 2-way

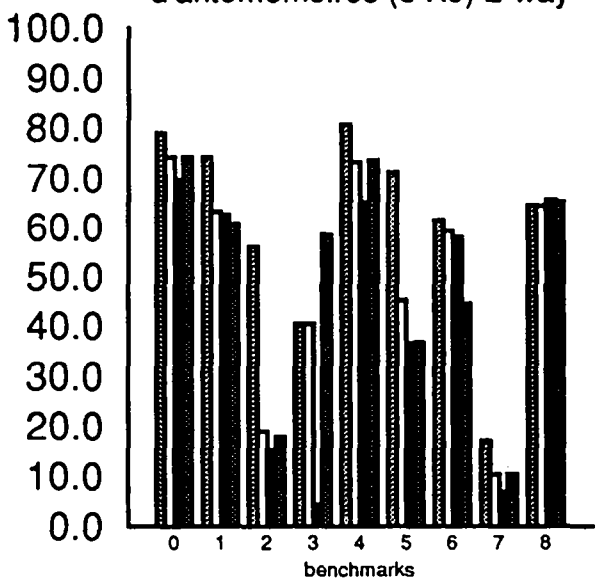
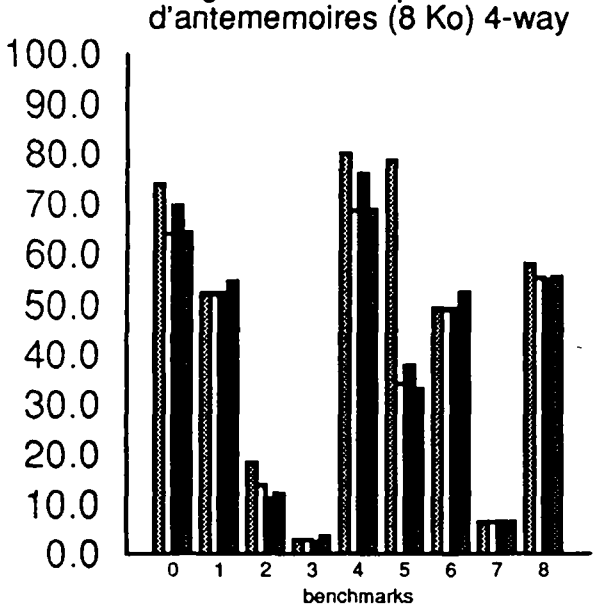


Figure 18: Comparaisons d'antememoires (8 Ko) 4-way



LEGENDE

Benchmarks	Organisations
0 cptc	SEP
1 aller	UNI
2 poisson2	SU
3 matmul1	MIX
4 cache	
5 strassen	
6 linpackud	
7 poisson1	
8 linpackrd	

A Traces

A.1 Générateur de traces

Un des plus importants programmes du système est le système d'exploitation. Il s'occupe de gérer les différentes ressources du microprocesseur et les différents processus utilisateurs. Il est généralement spécifique à une famille de microprocesseurs compatibles.

Afin de générer le plus correctement possible une trace, il faut que les principaux supports de ce système d'exploitation soient simulés [AGAR88], [AGAR89], [HOLL92]. En particulier ceux qui correspondent à la gestion des processus utilisateurs, ainsi que de la mémoire et des entrées-sorties. Pour avoir des traces représentatives, il faut donc prendre en compte les intérruptions et les changements de contexte.

Les traces utilisées sont simulés par le générateur de traces SPARCsim (SPARC Architectural Simulator) [SPARC90]. Ces traces proviennent de différents programmes.

A.2 Description des traces

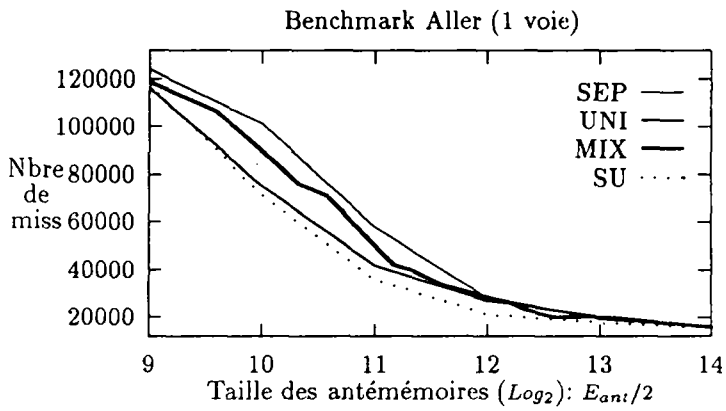
Origine et contenu des programmes utilisés pour générer les différentes traces:

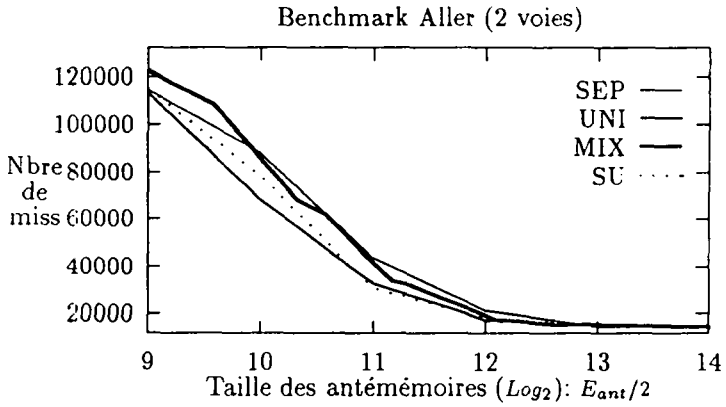
1. CPTC (écrit en Pascal): il génère à l'exécution sa traduction en langage C.
2. CACHE (écrit en C++): c'est notre simulateur d'antémémoires.
3. STRASSEN (écrit en C): multiplication de deux matrices suivant l'algorithme de Strassen.
4. POISSON (écrit en C): résolution selon la méthode de Poisson.
5. MULMAT (écrit en FORTRAN): multiplication de matrices creuses.
6. LINPACK (écrit en FORTRAN): résolution d'un système linéaire.

B Graphes

Nous allons donner quelques graphes de comparaison pour les différentes organisations d'antémémoires.

Figure 19 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





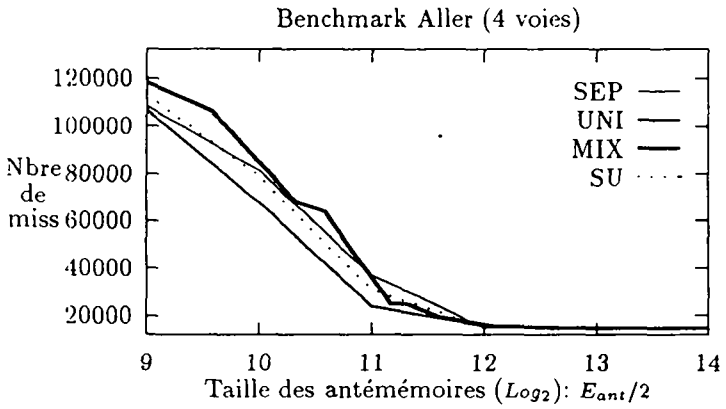
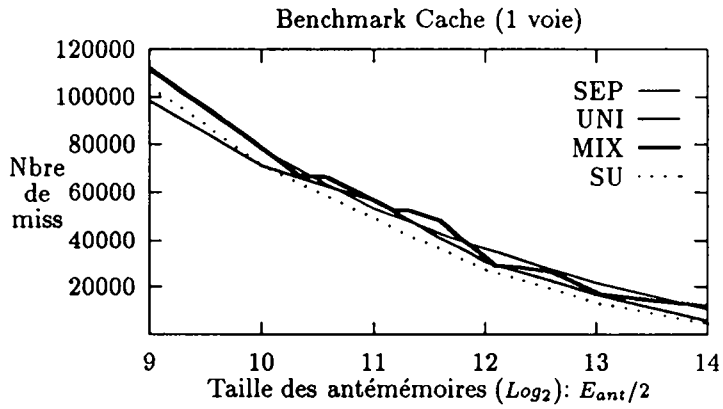
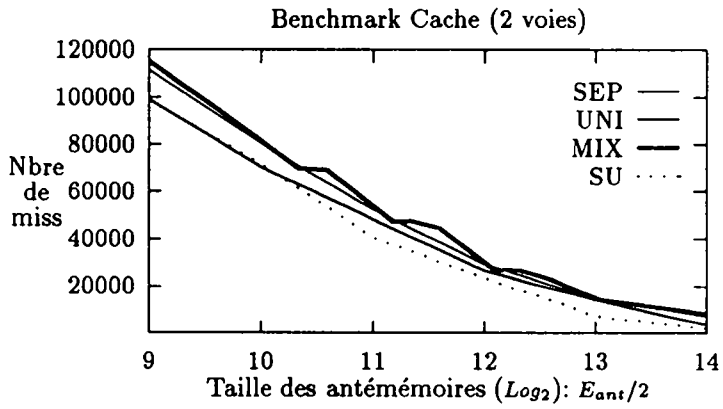


Figure 20 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





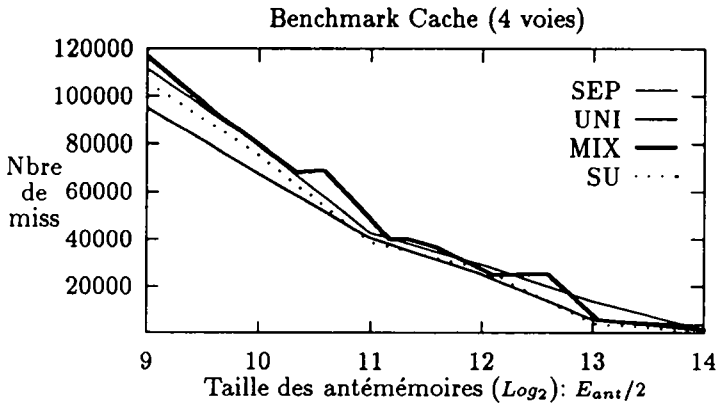
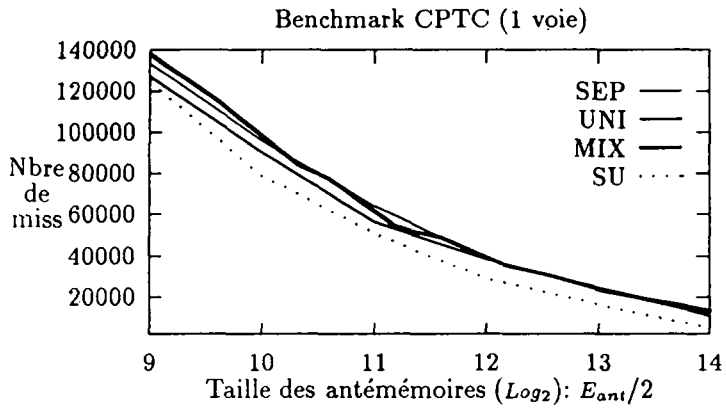
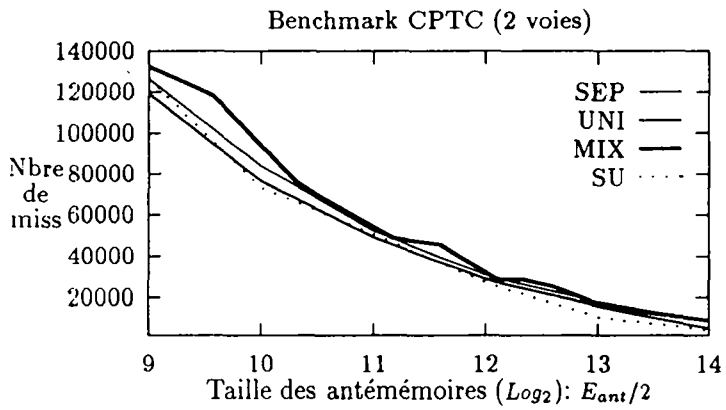


Figure 21 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





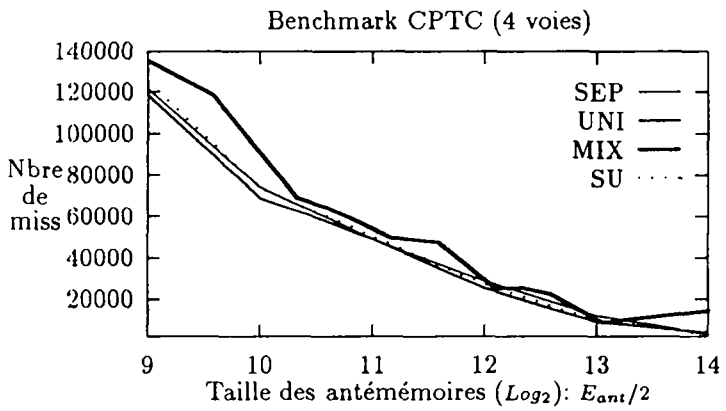
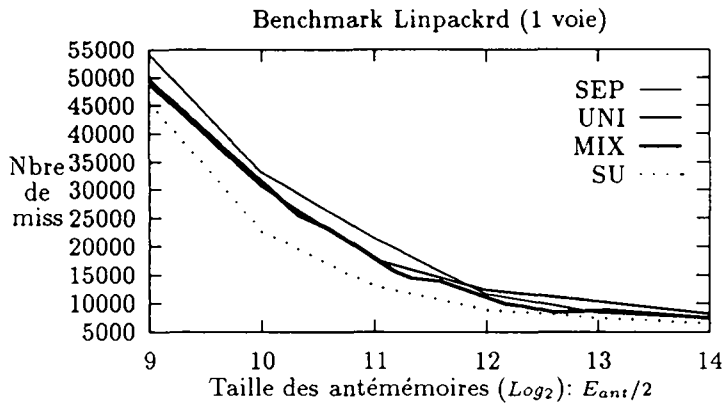
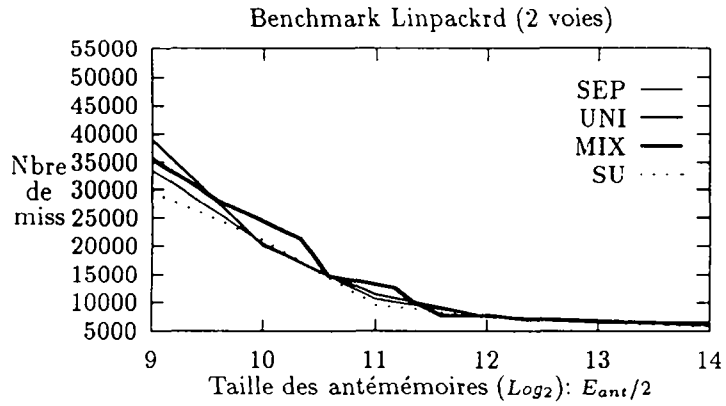


Figure 22 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





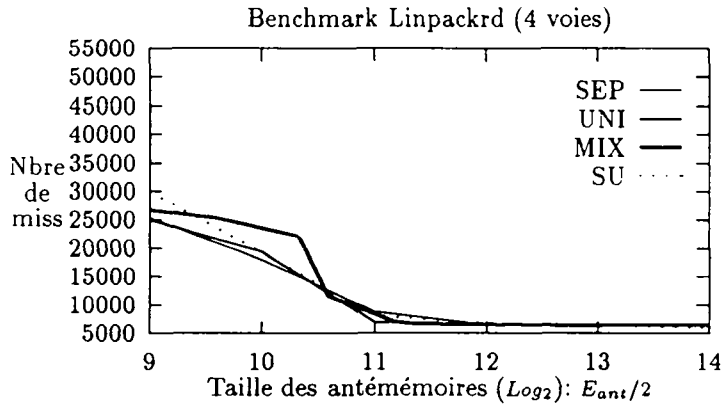
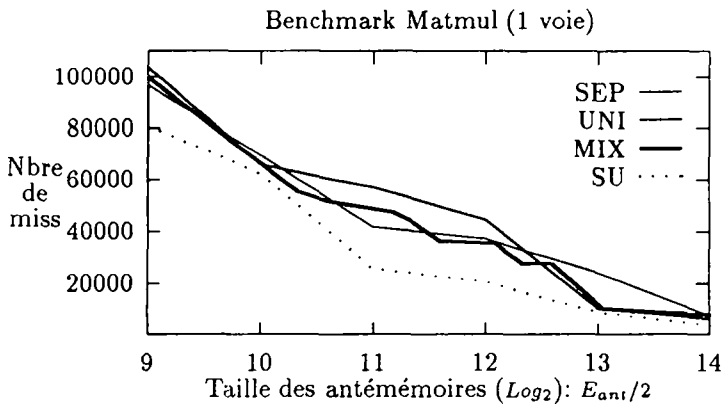
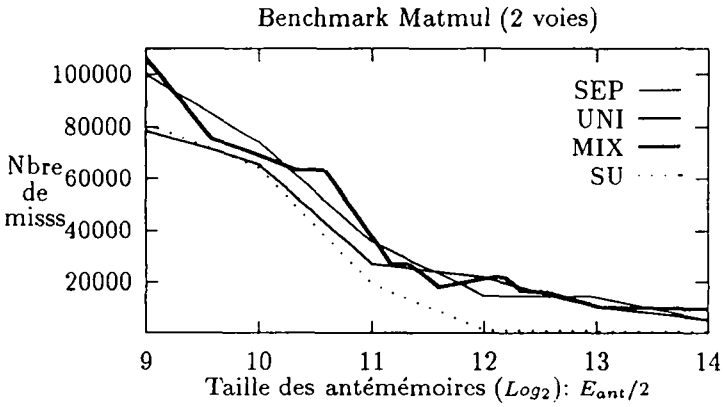


Figure 23 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





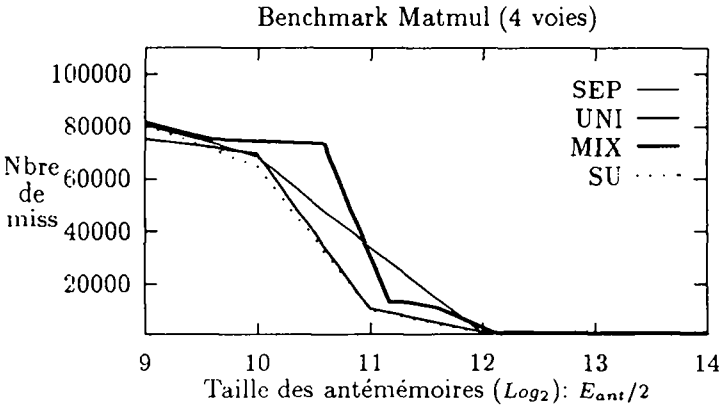
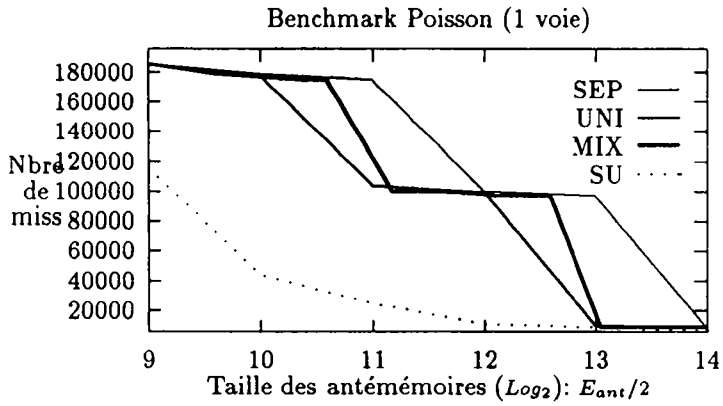
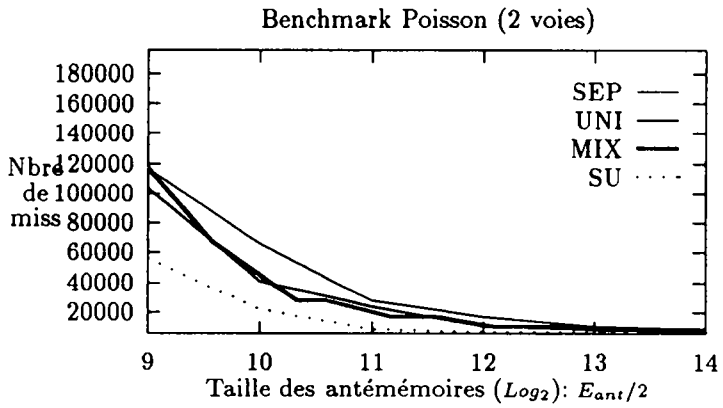


Figure 24 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.





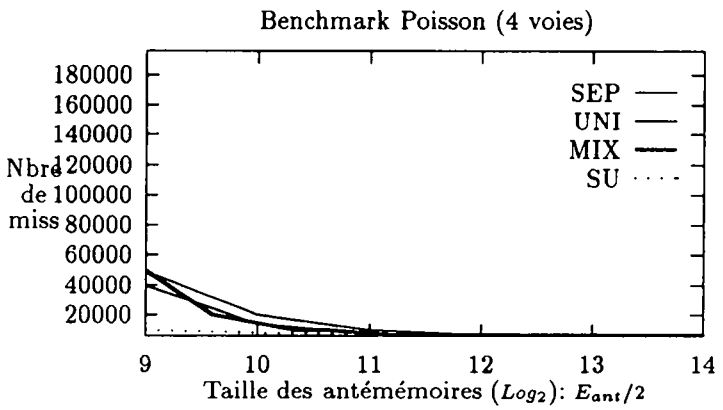
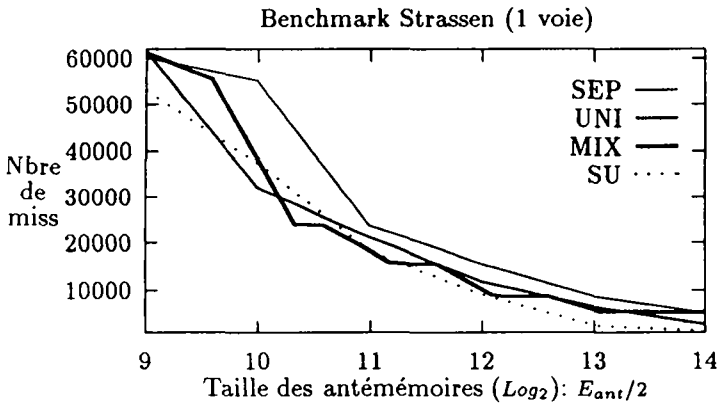
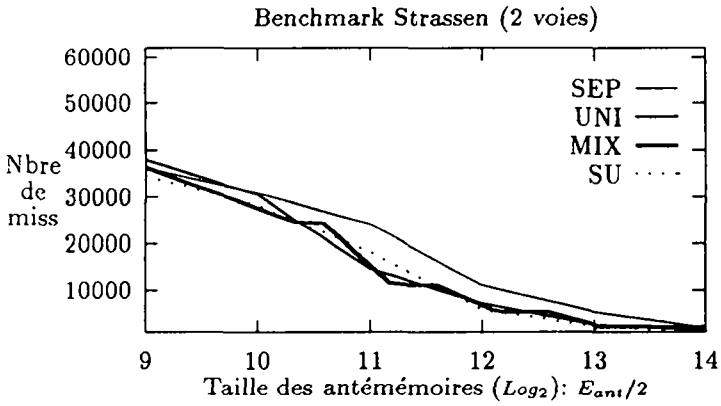
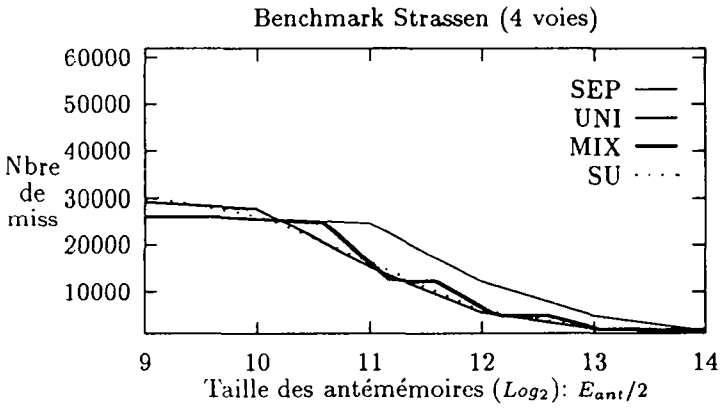


Figure 25 : Comparaison d'organisations d'antémémoires. Paramètre fixé: $L = 64$ o.







Bibliographie

[AGAR88] A. Agarwal, J. Hennessy, M. Horowitz, "Cache performance of operating system and multiprogramming workloads", ACM TOCS, novembre 1988.

[AGAR89] A. Agarwal, "Analysis of cache performance for operating systems and multiprogramming", édition Kluwer Academic Publishers, 1989.

[DEC21064] "Introduction to designing a system with the DECchip 21064 microprocessor", Digital Equipment Corporation, avril 1992.

[ETIEM91] D. Etiemble, "Architectures des processeurs RISC", édition A. Colin, 1991.

[FARR89] M.K. Farrens, A.R. Pleszkun, "Improving performance of small on-chip instructions cache", ISCA, juin 1989.

[HILL87] M.D. Hill, "Aspects of cache memory and instruction buffer performance", Ph.D de l'université de Californie, 1987.

[HILL88] M.D Hill, "A case for direct-mapped caches", IEE computer, décembre 1988.

[HILL89] M.D. Hill, A.J. Smith, "Evaluating associativity in CPU caches", IEEE transactions, décembre 1989.

[HOLL92] M.A. Holliday, C.S. Ellis, "Accuracy of memory reference traces of parallel computations in trace-driven simulation", IEEE transactions, janvier 1992.

[IBM RS6000] "IBM RISC system/6000 technology", IBM Corporation.

[IBMpower] "IBM POWER processor architecture version 1.52", IBM Corporation.

[i860] "i860: 64-bit microprocessor hardware reference manual", intel, 1990.

[JOUPPI90] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", ISCA, mai 1990.

[KOHN89] L. Kohn, S.W. Fu, "A 1,000,000 transistor microprocessor", IEEE, février 1989.

[MATI89] R.E. Matick, "Functionnal cache chip for improved system performance", IBM journal Research and developpement, janvier 1989.

[MET90] C. Metairie, N. Poliau, "C++: utilisation d'un langage orienté objet", édition Eyrolles, 1990.

[MIPS R4000] MIPS R4000User's Manual 1991.

[PAT90] J.L. Hennessy, D.A. Patterson, "Computer architecture, a quantitative approach", édition Morgan Kaufmann, 1990.

[PNEV90] D.N. Pnevmatikatos, M.D. Hill, "Cache performance of the integer SPEC benchmarks on a RISC", Compute architecture news, juin 1990.

[PRZY88] S.A. Przybylski, M. Horowitz, J. Hennessy, "Performance tradeoffs in cache design", ISCA, mai 1988.

[PRZY89] S.A. Przybylski, M. Horowitz, J. Hennessy, "Characteristics of performance optimal multi-level cache hierarchies", ISCA, mai 1898.

[PRZY90] S. A. Przybylski, "Cache and memory hierarchy design: a performance-directed approach", édition Morgan Kaufmann, Avril 1990.

[PUZAK85] T. R. Puzak, "Cache memory design", Ph.D de l'université du Massachusetts, 1985.

[SMITH82] A. J. Smith, "Cache memories", ACM Computing Surveys, septembre 1982.

[SMITH87] A. J. Smith, "Line size choice for CPU cache memories", IEEE transactions, septembre 1987.

[SMITH91] A. J. Smith, "Second Bibliography on Cache Memories", Computer Architecture News, 1991.

[SPARC90] Le manuel du SPARCsim, édition Sun microsystems, décembre 1990.

[SPARC91] The SPARCtechnical Papers, édition Springer-Verlag, 1991.

[STONE87] H. S. Stone, "High-performance computer architecture", édition Addison-Wesley Publishing Company, 1987.

[STREC83] W.D. Strecker, "Transient behavior of cache memories", ACM transactions, novembre 1983.

[superSPARC] "TMS390Z50: integrated sparc processor", Texas Instrument, 1992.

[Cache SPARC] "TMS390Z55: cache controller", Texas Instrument, 1992.

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 670 UN RESEAU SYSTOLIQUE INTEGRE POUR LA CORRECTION DE FAUTES DE FRAPPE
Dominique LAVENIER
Juillet 1992, 120 pages.

PI 671 EARLY WARNING OF SLIGHT CHANGES IN SYSTEMS AND PLANTS WITH APPLICATION TO CONDITION BASED MAINTENANCE
Qinghua ZHANG, Michèle BASSEVILLE, Albert BENVENISTE
Juillet 1992, 32 pages.

PI 672 ORDRES REPRESENTABLES PAR DES TRANSLATIONS DE SEGMENTS DANS LE PLAN
Vincent BOUCHITTE, Roland JEGOU, JeanXavier RAMPON
Juillet 1992, 8 pages.

PI 673 AN EXCEPTION HANDLING MECHANISM FOR PARALLEL OBJECT-ORIENTED PROGRAMMING
Valérie ISSARNY
Août 1992, 36 pages.

PI 674 A CALCULUS OF GAMMA PROGRAMS
Chris HANKIN, Daniel LE METAYER, David SANDS
Juillet 1992, 32 pages.

PI 675 EVALUATION DES PERFORMANCES D'UN NOYAU DE SIMULATION REPARTIE
Philippe INGELS, Carlos MAZIERO
Septembre 1992, 36 pages.

PI 676 FONT METRICS
Jacques ANDRE
Septembre 1992, 20 pages.

PI 677 GRIF ET LES INDEX ELECTRONIQUES
Hélène RICHY
Septembre 1992, 40 pages.

PI 678 ETUDE DE QUELQUES ORGANISATIONS D'ANTEMEMOIRES
Nathalie DRACH, André SEZNEC
Octobre 1992, 44 pages.

ISSN 0249-6399